Quantum Data 881/882 SDK 1.0

Programmer's Guide

quantumdata

Quantum Data 881/882 SDK 1.0 Programmer's Guide

Revision A2 (2006 July)

Updates to this document are available at http://www.quantumdata.com/support/downloads/.

Copyright 2006 Quantum Data. All rights reserved.

The information in this document is provided for use by our customers and may not be incorporated into other products or publications without the expressed written consent of Quantum Data. Quantum Data reserves the right to make changes to its products to improve performance, reliability, productibility, and (or) marketability. Information furnished by Quantum Data is believed to be accurate and reliable. However, no responsibility is assumed by Quantum Data for its use.

Contents

Chapter 1 Creating Images

Overview 2

Overview of image development process 2

Setting up your development environment 3

Tutorial: Creating hello.o image 5

Creating implementation file 5

Compiling image 7

Installing image 8

Loading image 10

Chapter 2 Graphics Functions

Functions by type 14

Functions by name 22

Color set definitions 171

Color definitions 176

Chapter 3 Image Examples

PulseBar 204

SMPTEBar 205

QuartBox 206

BurstTCE 207

Diamond 208

Samples 209

1 Creating Images

Topics in this chapter:

- Overview
- Setting up your development environment
- Tutorial: Creating hello.o image

Overview

The SDK provides Application Programming Interfaces (APIs) to functions implemented in the generator firmware for adding custom images to your generator. The standard images provided with the generator use these same APIs. For more information about the APIs, see Chapter 2, "Graphics Functions."

It is possible to use generator commands to create images (.cmd files). However, command-based images are executed through an interpreter, and render slower than compiled binary files (.o files). The SDK can be used to create images that load faster, and that are more complex, than command-based images.

Overview of image development process

Creating custom images using the SDK is straight forward process. However, knowledge of C/C++ programming is necessary for creating complex images.

General steps for creating an image:

1. Install the SDK and setup your development environment.
2. Create an implementation (C++ source code) file using any source code editor.
3. Use compiler provided with SDK to build image object file (.o file).
4. Store the image object file at a location that the generator can access.

   You can use an FTP client to copy the file to the generator, or configure the generator to load the file from a file server.
5. Load the image with the generator.

   Images can be loaded using the generator control panel, or from a remote location by sending commands through a serial or telnet connection. You can also load images using a web browser to operate the virtual control panel of the generator.

   When the image is initially loaded, the .o file is dynamically linked with the generator.

Setting up your development environment

This section describe how to setup a computer to build images. Setting up your development environment involves creating a directory structure for your compiler, header files, implementation files and various executables on a WindowsNT/2000/XP Professional Pentium-based PC.

To test compiled images, the development PC should be connected to the same LAN as the generators, or connected directly (using a patch cable) with a generator.

To install the SDK:

- Copy the \882\sdk folder from the Resource CD (provided with your generator) to the root directory on your C: drive.

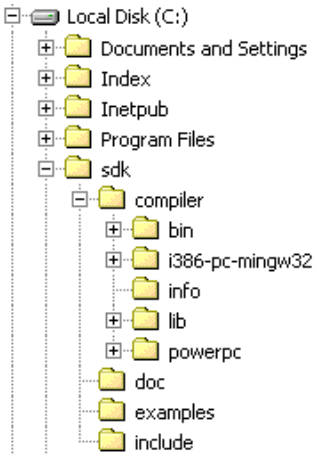
  The following folders and files are installed.

Folder	Contents
sdk\compiler	GNU GCC ANSI C compatible compiler 2.7.2. Version 2.7.2 of this compiler is recommended for creating images for the 882 generator. This compiler can be installed on a Windows-based PC to produce binary files for execution on PowerPC processors.
sdk\doc	This user manual.
sdk\examples	Source code for selected standard images (PulseBar, Diamond, SMPTEBar, QuartBox and Burst) provided with the generator. For descriptions of these images, see Chapter 3, "Image Examples." Source code for sample image (sample.cpp) that demonstrates most of the graphics-related functions supported by the SDK.
sdk\include	Header files: • **Gc.h** includes CoreGc.h, Lutstep.h, MathLib.h and provides all the function definitions for initialization, image version control, video handling, transparency, screen filling, drawing primitives, drawing composites, object filling, pattern filling, text rendering, font management, 2-D drawing, 3-D drawing, window management, bit manipulation and various utilities. Your implementation file needs to include only the Gc.h header file, which includes all the necessary header files. • **CoreGC.h** provides class definitions, platform register access, and fixed point functions. • **LutStep.h** provides masks, color assignments and tables. • **Mathlib.h** provides math functions including math.h function wrappers. • **Canvas.h** provides variable class definitions for manipulating primatives. • **Stdcolor.h** provides color identifiers. • **QD_types.h** provides Quantum Data specific variable type definitions. • **stdio.h**. Using this version of stdio.h is recommended.

To install the complier:

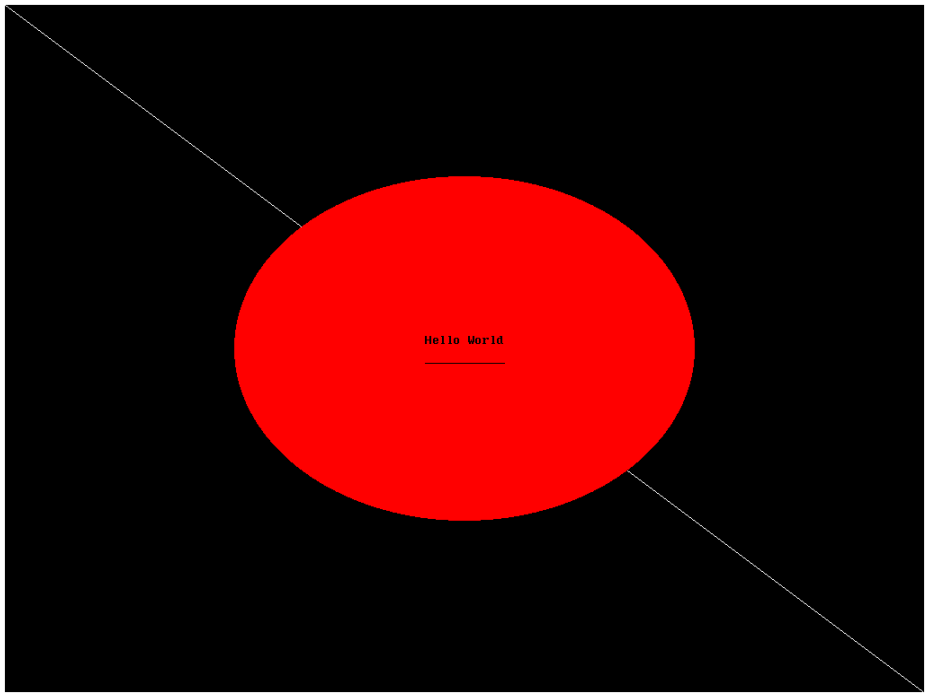
1. Open the c:\sdk\compiler folder.
2. Expand the SDK.ZIP to the current folder.

   The following folders are created.



Tutorial: Creating hello.o image

This section describes how to program, build, and use a simple image, which is shown below.



Source code for additional, more complex, images is available in the c:\sdk\examples folder. For descriptions of these images, see Chapter 3, “Image Examples.”

Creating implementation file

After setting up your development environment, the next step in the process of creating your own images is to write the implementation file (.cpp file). The implementation file for the hello image is provided with the SDK.

To review the implementation file:

1. Open the c:\sdk\examples\hello.cpp, which is listed below.

```
#include <gc.h>

/*****************************************************************
*
*  Render_hello
*
*  Usage:
*     Render_hello( TGC* gc );
*
```

```
*  Arguments:
*     gc - the graphics context being drawn on.
*
*  Returns:
*     bool completed - true if the image could be drawn.
*                    - false if it can't be drawn.
*
*****************************************************************/
//*** IMPORTANT: Image entry point naming convention ***
//
// The name of this function is created as follows:
//
//   - Take the name of this source file without the extension
//     and prepend "Render_".
//
// Failure to follow this naming convention for the image entry point
// will prevent the generator from recognizing the file as an image.
//
bool Render_hello( TGC* gc )
{
  INT32 w, h, x, y, strPixelWidth;

  // Indicate that there are no alternate image renditions
  gc->VertexToPoint( 0 );

  // Setup the lookup table:
  // Black, white and some basic colors
  gc->SetLut( BW | HUERGB );

  // Clear the active screen area
  gc->ZeroActive();

  // Draw a white line from the top-left corner of the active area to
  // the bottom-right corner.
  // Also draw a border (rectangle) around the active screen area.
  // The TGC instance we are given has its parameters pre-configured
  // by the generator based on active video format.
  gc->Opaque( TStdColor::white, TStdColor::black );
  gc->DrawLine( gc->left, gc->top, gc->width, gc->height );
  gc->DrawRect( gc->width, gc->height, gc->left, gc->top );

  //-----------------------------------------------------
  // Draw a red filled oval centered on the screen
  //-----------------------------------------------------

  // Calculate the oval dimensions and position
  w = gc->width / 2;
  h = gc->height / 2;
  x = gc->centerleft - w/2;
  y = gc->centertop - h/2;

  // Select color
  gc->Transparent( TStdColor::red );
```

```
    // Render oval
    gc->FillOval( w, h, x, y );

    // Render some text centered in the middle of the screen
    // and draw a line under the string.

    // Create text string
    const char* txt = "Hello World!";

    // Select font and color (black)
    gc->SelectFont( "SYS16" );
    gc->Transparent( TStdColor::black );

    // Calculate location of string
    strPixelWidth = gc->GetWidth( txt );

    // Width of string in pixels
    x = (gc->width - strPixelWidth) / 2;
    y = (gc->height - gc->CharHigh()) / 2;

    // Draw string
    gc->DrawString( x, y, txt );

    // Draw underline
    gc->DrawLine( x, y + gc->GetDescent(), x + strPixelWidth, y +
    gc->GetDescent() );

    // Return "true" to indicate success.
    return true;
}
```

Compiling image

After creating the implementation file, you need to compile it to a .o (object) file, which the generator can render.

The c:\sdk\examples\folder contains an objcompile.bat file which is setup to compile images.

To compile the source file:

1. Configure your folders such that the SDK folder is directly under the C drive.

   This is important because the batch file used to compile the image implementation files sets the system path in accordance with this structure.

2. Open up a DOS command window.

3. Change to the directory that contains the source file you want to compile using the DOS CD command. The hello.cpp file is in the c:\sdk\examples directory.

4. Enter the following command:

   ```
   objcompile.bat sourceFileName
   ```

   Where *sourceFileName* is the name of the source file you want to compile. For example to compile the hello.o image you would enter:

   ```
   objcompile.bat hello
   ```

   If the image compiled successfully, the system responds with:

   ```
   !! SUCCESS !!
   ```

5. Verify that the hello.o file is present in the directory by enter the DIR command.
6. (Optional) The generator user interface uses the name of the .o file to identify the image. If you want to use a different name, rename the .o file.

Installing image

After building the .o file, you need to copy the file to a location that the generator can access. Images can be rendered either from the generator itself or from a file server.

Note: *This section assumes that you have setup a file server, and that the generator is connected to the same LAN as your development workstation. For information about completing these tasks, see the 882 User's Guide which is available on the Resource CD.*

You can use any FTP client to transfer files to the generator, or use the browser-based file transfer utility provided with the generator.

To transfer files to the generator using FTP:

1. Open up a command window.
2. Change to the directory that contains the image (.o file) you wish to transfer to the generator (for example, c:\sdk\examples).
3. Enter the FTP command on the DOS command line at the prompt.

   ```
   C:\sdk\examples>ftp address
   ```

   Where *address* is the IP address of the generator.

   The system responses with:

   ```
   Connected to 206.135.215.168.
   220 VxWorks FTP server (VxWorks 5.4.2) ready.
   ```

4. When prompted for a user name, press Enter without typing a user name.

   ```
   User (206.135.215.168:(none)):
   331 Password required
   ```

5. When prompted for a password, press Enter without typing a password. No password is required.

```
Password:
230 User logged in
```

6. Enter bin to specify a binary file type.

```
ftp>bin
200 Type set to I, binary mode
```

7. Verify that you are at the proper storage device. For example if you want to install the image in the flash memory (tffs0) you would enter:

```
pwd
```

The system responds with:

```
Current directory is "/tffs0/library"
```

8. Enter the following command to change to the Images directory, which is where image files are stored.

```
ftp>cd library/images
```

9. Use the following commands to copy the hello.o file to the generator:

```
ftp>put hello.o
200 Port set okay
150 Opening BINARY mode data connection
226 Transfer complete
ftp: 1752 bytes sent in 0.00Seconds 1752000.00Kbytes/sec.
```

10. Terminate the FTP session with:

```
ftp>bye
```

To remove files from the generator:

1. Establish a telnet session with the generator.

Note: *You can also use an FTP client to remove files from the generator.*

```
C:\sdk\examples>telnet <IP address> // example: 206.135.215.168
```

The generator provides you with a prompt with the current directory.

2. Change directories to the directory where the file you want to delete resides:

```
\tffs0>cd library\images
```

3. List the files with the list query:

```
\tffs0\library\images>ls?
```

The generator lists the contents of the Images directory.

4. Enter the remove file command:

```
\tffs0\library\images>rm hello.o
```

List the contents to verify that the file has been deleted.

To copy files to a generator using a web browser:

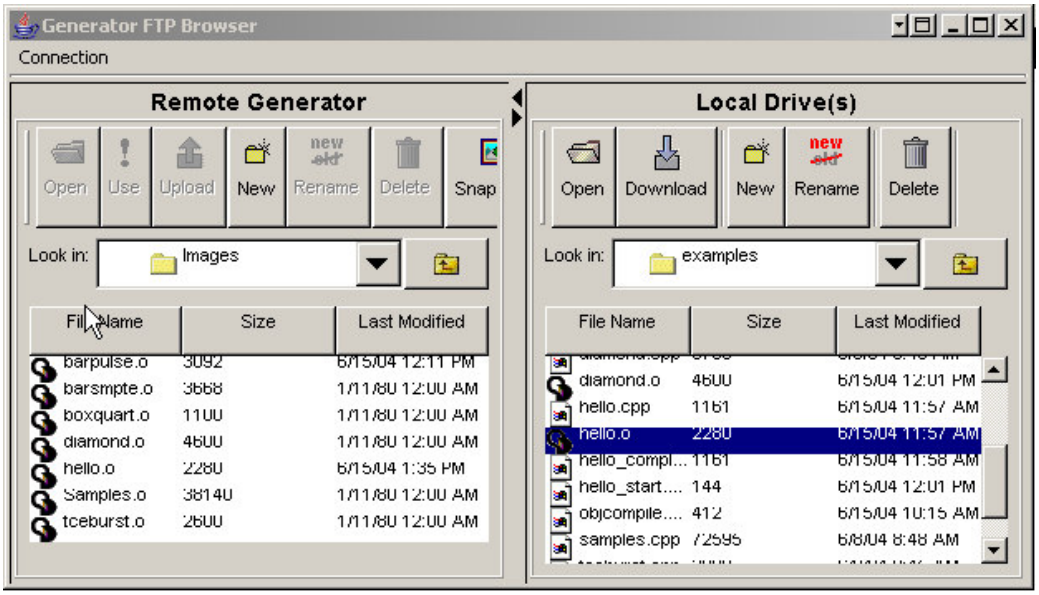
1. Open a web browser and enter the IP address of the generator to which you wish to transfer the image.

   The generator home page opens.

2. Click the Virtual Front Panel link.

   From the Options menu (upper left corner of virtual control panel), choose the Browse option.

   The FTP Browser opens.



3. In the Local Drive pane, choose the file you want to copy to the generator, and then click the Download icon.

Loading image

The generator can load images stored locally, on the generator, or images stored on a file server. Regardless of the location, you can use the control panel (or virtual control panel) or the command-line interface to load an image.

Before loading an image, connect the generator with a display, and choose a format supported by the display.

To load a local image using the control panel:

1. With the generator in the Browse mode, press the Home key.
2. Choose the FlashMem (local generator) option.

3. Open the Generator folder, and then open the Images folder.

   Available images are listed.

4. Use the + and - keys to scroll to the image you want to load, and then press the corresponding soft key to load the image.

To load a local image using the command-line interface:

1. Establish a serial or telnet session with the generator.
2. Set the image path parameter to the images directory.

   ```
   imgp /tffs0/library/images
   ```

3. Load and display the image using the following command string:

   ```
   imgl hello.o; imgu
   ```

2 Graphics Functions

Topics in this chapter:

- Functions by type
- Functions by name

Functions by type

Type (class)	Function	Description
2-D methods	FTransformX	Transforms a normalized coordinate into an equivalent machine unit on x axis.
	FTransformY	Transforms a normalized coordinate into an equivalent machine unit on y axis.
	SquareDX	Scales machine units on the y axis into another machine units on the x axis.
	SquareDY	Scales machine units on the x axis into another machine units on the y axis.
	TransformDX	Transforms a normalized distance into an equivalent distance in machine units along the x axis.
	TransformDY	Transforms a normalized distance into an equivalent distance in machine units along y axis.
	TransformX	Transforms a normalized coordinate into an equivalent machine unit on the x axis.
	TransformY	Transforms a normalized coordinate into an equivalent machine unit on the y axis.
	TranslateX	Translates the origin used by the transform and translate functions.
	TranslateY	Translates the origin used by the transform and translate functions.
3-D methods	CopyMatrix	Copies a 4x4 input matrix to a 4x4 output matrix.
	CopyVertex	Copies an input vertex list to an output vertex list.
	InitMatrix	Initializes memory representing a 4x4 matrix to the identity matrix.
	Perspec	Performs perspective transformations on a list of 3D vertices, mapping them to a list of 2D points.
	ResetMatrix	Clears the transformation matrix and resets it to the default values of the origin in upper left hand corner.
	Rotate	Multiplies the specified 4x4 transformation matrix by a rotation matrix created from the three angles given as input arguments.
	Scale	Multiplies the specified 4x4 transformation matrix by a scaling matrix created from the three scaling factors given as input arguments.
	Transform	Uses a 4x4 transformation matrix to transform (rotate, scale and translate) a list of 3D vertices.

Type (class)	Function	Description
	Translate	Multiplies the specified 4x4 transformation matrix by a translation matrix created from the three displacements given as input arguments.
Bitmap icons	GetPixel	Returns the value of the pixel at coordinates (x,y).
	GetRect	Copies pixels contained in a rectangular area of the screen into a specified packed pixel array.
	MoveRect	Copies from one rectangular region of the screen to another.
	PixelExpand	Moves the contents of the specified packed pixel array into a specified rectangular area on the screen.
	Class	Writes the specified pixel value to the specified coordinates.
	PutRect	Moves the contents of the specified packed pixel array into a specified rectangular area on the screen.
	ZoomRect	Expands or shrinks a source rectangle on the screen to fit dimensions of a destination rectangle that is on the screen.
Color	GetEdgeColor	Sets and creates the edge color for each transition sample at the sine-square function.
	GetEdgeColorMany	Returns the number of samples at the transition edge between colors.
	GetLevelMax	Gets the maximum luminance level.
	PokeBlackColors	Sets the intensities of red, green, and blue primaries of the currently selected color code in an analog look-up-table to zero level.
	PokeCode	Sets DAC code directly at the current color address of the digital look-up-table.
	PokeColor	Directly loads a DAC color code into the look-up table (LUT).
	PokeLevelRGB	Sets the intensities of red, green, and blue primaries of the currently selected color code in an analog look-up-table (LUT).
	RepPixel	Replicates a pixel throughout a 32-bit integer.
	SelectColor	Sets the current color code for the look-up table (LUT).
	SetCode	Set specified code at the current color address of the digital look-up- table.
	SetColorNamed	Sets the current color name used for drawing.
	SetEdgeColorNamed	Sets and creates edge colors for rendering anti-aliased TV images.

Type (class)	Function	Description
	SetLevelGray	Sets all primaries of the current color to a specified level of the analog look-up-table (LUT).
	SetLevelGrayPercent	Sets the intensity level of the primaries of the current color in the analog look-up-table (LUT).
	SetLevelMax	Sets the maximum luminance level.
	SetLevelRGB	Sets the intensities of red, green, and blue primaries of the currently selected color code in an analog look-up-table (LUT).
	SetLut	Initializes both digital and analog look-up tables (LUTs) given a colorset token.
	UnRepPixel	Unreplicates pixel throughout a 32-bit integer.
Drawing composites	DrawCenterMark	Draws a cross in the center of the screen.
	DrawCross	Draws a cross in the center of the screen using the current COLOR1.
	DrawFormat	Fills the fixed rectangle using COLOR0 and draws the outline of the rectangle along with the information about the used format using COLOR1.
	DrawGrid	Draws grids of different types.
	DrawGrill	Draws a grill in a specified window.
	DrawLimits	Draws screen limits using the current COLOR1.
	DrawStep	Fills the fixed rectangle using COLOR0 and draws the outline of the rectangle along with the step number of the image in the sequence using COLOR1.
	DrawTBox	Fills the fixed rectangle using COLOR0 and draws the outline of the rectangle along with the information using COLOR1.
Drawing primitives	DottedOval	Draws the dotted outline of an ellipse given the minimum enclosing rectangle in which the ellipse is inscribed and the spacing angle between the dots.
	DrawLine	Uses Bresenham's algorithm to draw a line from the starting point to the ending point.
	DrawOval	Draws the outline of an ellipse given the minimum enclosing rectangle in which the ellipse is inscribed.
	DrawPoint	Draws a single pixel at the specified position.
	DrawPolyLine	Draws multiple lines.
	DrawRect	Draws the outline of a rectangle.
	DrawTriangle	Draws the outline of a triangle.

Type (class)	Function	Description
Filling primitives with solid	CurtainFillRect	Fills a specified rectangular area by copying either the top row or left column into the balance of the rectangle.
	EncloseFillRect	Draws a filled-in rectangle.
	FillOval	Draws an ellipse that is solid-filled with the current COLOR1.
	FillPolygon	Fills a polygon given a list of points representing the vertices of the polygon.
	FillRect	Draws a rectangle that is solid-filled with the current COLOR1.
	FillTriangle	Draws a triangle that is solid-filled with the current COLOR1.
	FrameOval	Solid-fills an ellipse-shaped frame with the current COLOR1.
	FrameRect	Solid-fills a rectangular frame with the current COLOR1.
Fixed point functions	FixToInt16	Converts an array of fixed-pont numbers to short integers.
	FixToInt32	Converts an array of fixed-point numbers to long integers.
	Int16ToFix	Converts an array of short integers to fixed- point format.
	Int32ToFix	Converts an array of long integers to fixed-point format.
	VertexToPoint	Converts a vertex list to a point list by copying the integer parts of the x and y coordinates from the vertex list to the point list.
Font management	CharHigh	Returns the character height (in pixels) for the current font.
	CharWideImage	Calculates and returns the image width (in pixels) of the specified ASCII character.
	CharWideMax	Returns the width (in pixels) of the widest character in the current font.
	GetDescent	Returns the value of the descent parameter for the current font.
	GetFirstCh	Returns the ASCII character code for the first character present in the current font.
	GetFontMax	Returns the maximum number of fonts that can be installed simultaneously.
	GetLastCh	Returns the ASCII character code for the last character present in the current font.
	GetLeading	Returns the leading value for the current font.
	SelectFont	Selects the font identified by an index or by name.

Type (class)	Function	Description
	SelectFontNamed	Selects the font identified by name.
Image version	SetVersions	Sets quantity of image versions (all images must call this function).
Patterns	GetPatnMax	Returns the maximum number of patterns that can be installed at any one time.
	InstallPatn	Installs a pattern in the pattern table.
	PatnFillOval	Fills an ellipse with a pattern.
	PatnFillPolygon	Fills a polygon with a pattern given a list of points representing the vertices of the polygon.
	PatnFillRect	Fills a rectangle with a pattern.
	PatnFillTriangle	Fills a triangle with a pattern.
	PatnFrameOval	Fills an ellipse-shaped frame with a pattern.
	PatnFrameRect	Fills a rectangular-shaped frame with a pattern.
	RemovePatn	Installs 0s 16 x16 bitmap into the last slot (USER_PATN) in the pattern table.
	SelectPatn	Selects the pattern identified by the specified index or name.
Register access	Get_Pmask	Returns the value of the plane mask (TMS34010 PMASK register).
	Get_ppop	Returns the code for the current pixel processing operation (the PPOP field in the TMS34010's CONTROL register).
	GetColor0	Retrieves the 32-bit color value stored at LUT location 0.
	GetColor1	Retrieves the 32-bit color value stored at LUT location 1.
	GetPsize	Returns the pixel depth currently being used by the generator.
	GetTransp	Returns the current state of the transparency enable bit in the generator's control register.
	Set_Pmask	Loads a generator's plane mask (PMASK) register with specified value.
	Set_ppop	Defines the pixel processing operation for subsequent drawing operations.
	SetColor0	Sets look-up table (LUT) color 0 to a specific pixel value.
	SetColor1	Sets look-up table (LUT) color 1 to a specific pixel value.
	TranspOFF	Disables transparency, and the result of a subsequent pixel operation is written to the destination regardless of its value.

Type (class)	Function	Description
	TranspON	Clears the T (transparency) bit in the generator's rendering color registers.
Screen fills	BlackFillVuport	Sets the entire extent of the active portion of memory to the color 0x000000 (black).
	ColorFillVuport	Sets the entire extent of the active portion of memory to specified color.
	GroundFillVuport	Sets the entire extent of the active portion of memory to the current foreground color.
	WipeActive	See GroundFillVuport.
	ZeroActive	See BlackFillVuport.
	QD_FillActive	See ColorFillVuport().
Text	AddWordSpace	Changes the horizontal spacing between words by an amount n.
	AddMoreSpace	Changes the horizontal micro-spacing between words by an amount n.
	AddTextSpace	Changes the horizontal spacing between characters by an amount n.
	CharFillRect	Fills the entire active region of the display screen with a single character.
	DrawChar	Draws a single bitmapped character to the specified screen coordinates.
	DrawString	Draws a string of characters to the specified position on the screen.
	GetAscent	Returns the value of the ascent parameter for the current font.
	GetMoreSpace	Gets the value of the count after drawing the text.
	GetTextSpace	Gets the horizontal spacing between characters.
	GetWidth	Computes and returns the width in pixels of the specified ASCII string if drawn using the currently selected text font.
	GetWordSpace	Gets the horizontal spacing between words.
Transparency	Opaque	Sets the foreground color and the background color.
	Transparent	Sets the foreground color.
Utility	Boxes	Draws boxes of different types.
Windows	CloseVuport	Closes the viewport identified by the specified index.
	CopyVuport	Copies all attributes of the source viewport to the destination viewport.
	CurrentVuport	Returns the index of the current viewport as an int32.

Type (class)	Function	Description
	GetVuportMax	Returns the maximum number of viewports that can be opened at any one time.
	MoveVuport	Moves the viewport to a new position.
	OpenVuport	Opens a new viewport and returns an index that is used to identify the viewport in subsequent transactions.
	RemoveVuport	Removes (closes) the viewport identified by the specified index.
	SelectVuport	Selects the viewport identified by the specified index.
	SetClipRect	Sets the size and position of the clipping rectangle for subsequent drawing operations.
	SetOrigin	Locates the x-y origin for the current viewport at the specified position, given in terms of displacements (x, y) from the top left corner of the viewport.
	SizeVuport	Changes the width and height of the selected viewport.

Functions by name

AddMoreSpace

Class Text

Description Changes the horizontal micro-spacing between words by an amount *n*. It is actually a count of maximum value equals the number of spaces between the words, which is decreased by one each time it is used. Assume a text of five words (four spaces between its words). For n=0, it means add nothing in between the words. For n=1, it means add one pixel in the first space (between the first and the second word). For n=4, add one pixel in each space. For n=5, just add one pixel between the words.

Syntax AddMoreSpace(int32 *n*)

n
Amount of horizontal space between words.

Example This example draws an image that has text in which the space between some words is increasing.

```
void Test_AddMoreSpace( TGC* gc )
{
  INT16 x, y, i;
  char  *s;
  INT32 os = gc->GetMoreSpace();

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** set the active color
  gc->Transparent( TStdColor::white );

  gc->SelectFont( "OPIX9" );

  s = "Note increasing spaces between words.";

  for( i = 0, y = 0; i < 20; ++i )
  {
    gc->AddMoreSpace(i);
    x = 320 - gc->GetWidth(s)/2;
    y += gc->CharHigh();
    gc->DrawText( x, y, s );
  }

  gc->AddMoreSpace( os );
}
```

AddTextSpace

Class Text

Description Changes the horizontal spacing between characters by an amount *n*. Once text spacing is modified by the AddTextSpace() function, the spacing increment remains in effect within the viewport until this function is called again.

Syntax AddTextSpace(int32 *n*)

n

Amount of horizontal space to be added between characters. Associated with each text font is a default spacing between a character and the character to its right. When a string of characters is drawn to the screen, *n* is added to the default spacing between characters defined in the data structure for the current font. Argument *n* is specified in multiples of the pixel width; it can be positive or negative, depending on whether you wish to increase or decrease the spacing.

Example This example draws an image that has a text in which the space between the characters is increasing/decreasing.

```
void Test_AddTextSpace( TGC* gc )
{
  INT16 x, y, i;
  char  *s;
  INT32 os = gc->GetTextSpace();

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** set the active color
  gc->Transparent( TStdColor::white );

  gc->SelectFont( "OPIX9" );

  s = "Note increasing spaces between characters.";

  for( i = -8, y = 0; i < 20; ++i )
  {
    gc->AddTextSpace( i );
    x = 320 - gc->GetWidth(s)/2;
    y += gc->CharHigh();
    gc->DrawText( x, y, s );
  }

  gc->AddTextSpace( os );
}
```

AddWordSpace

Class Text

Description Changes the horizontal spacing between words by an amount *n*.

Syntax AddWordSpace(int32 *n*)

n

Amount of horizontal space to be added between words. Associated with each text font is a default spacing between a word and the word to its right. When a string of words is drawn to the screen, *n* is added to the default spacing between words defined in the data structure for the current font. Argument *n* is specified in multiples of the pixel width; it can be positive or negative, depending on whether you wish to increase or decrease the spacing.

Example This example draws an image that has a text in which the space between words is increasing.

```
void Test_AddWordSpace( TGC* gc )
{
  INT32 x, y, i;
  char  *s;
  INT32 os = gc->GetWordSpace();

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** set the active color
  gc->Transparent( TStdColor::white );

  gc->SelectFont( "OPIX9" );

  s = "Note increasing spaces between words.";

  for( i = 0, y = 0; i < 20; ++i )
  {
    gc->AddWordSpace(i);
    x = 320 - gc->GetWidth(s)/2;
    y += gc->CharHigh();
    gc->DrawText( x, y, s );
  }

  gc->AddWordSpace( os );
}
```

BlackFillVuport

Class Screen fills

Description Sets the entire extent of the active portion of memory to the color 0x000000 (black).

Syntax BlackFillVuport(void)

Example This example draws an image with a black background.

```
void Test_BlackFillVuport( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Fill the screen with black
  gc->BlackFillVuport( );
}
```

Boxes

Class Utility

Description Draws boxes of different types.

Syntax Boxes(short* *xbox*, short* *ybox*, long *density*, long *even*)

xbox, ybox
Pointers to box quantity variables.

density
0, 1, 2

even
<>0 then make odd results next higher even if odd.

W:H	W/H	Even==0			Even<>0		
		d=0 X:Y	d=1 X:Y	d=2 X:Y	d=0 X:Y	d=1 X:Y	d=2 X:Y
16:9	1.777...	16:10	16:9	32:18	16:10	16:10	32:18
5:3	1.666...	16:10	15:9	30:18	16:10	16:10	30:18
4:3	1.333...	14:10	16:12	32:24	14:10	16:12	32:24
1:1	1.000...	10:10	12:12	24:24	10:10	12:12	24:24
3:4	0.750	10:14	12:16	24:32	10:14	12:16	24:32

Example This example draws an image that has a dotted 24 x 32 hatch grid.

```
void Test_Boxes( TGC* gc )
{
  INT32 xbox, ybox;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->Boxes( &xbox, &ybox, 2, -1 );
  gc->Transparent( TStdColor::green );
  gc->DrawGrid( (INT32)xbox, (INT32)ybox, 31 );
}
```

CharFillRect

Class Text

Description Fills the entire active region of the display screen with a single character.

Syntax CharFillRect(int32 *width*, int32 *height*, int32 *left*, int32 *top*, int *spacex32*, int32 *spacey*, int32 *marginx*, int32 *marginy*, int32 *code*)

width
Width of enclosing rectangle.

height
Height of enclosing rectangle.

left
Coordinate of the left edge of the enclosing rectangle.

top
Coordinate of the top edge of the enclosing rectangle.

spacex
Horizontal space between character images.

spacey
Vertical space between character images.

marginx
Minimum space between character images and the right and left sides of the enclosing rectangle.

marginy
Minimum space between character images and the top and bottom sides of the enclosing rectangle.

code
Character within current font to use (0-255).

Example This example draws an image that has a rectangular region filled with the red Cs and another region filled with the '#' character.

```
void Test_CharFillRect( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the color to be red
```

```
  gc->Transparent( TStdColor::red );

  gc->SelectFont( "BIG30" );

  //*** Fill a rectangle with the character C
  gc->CharFillRect( 100, 80, 10, 20, 2, 1, 0, 0, 'C' );

  //*** Fill another rectangle with '#'
  gc->SelectFont( "sys16" );
  gc->FillRect( 200, 200, 200, 200 );
  gc->Opaque( TStdColor::white, TStdColor::black );
  gc->CharFillRect( 200, 200, 200, 200, 5, 5, 5, 5, '#' );
}
```

CharHigh

Class Font management

Description Returns the character height (in pixels) for the current font.

The character height is defined as the vertical distance between two adjacent rows of text, as measured from the two baselines. The character height is calculated as the sum of three quantities: ascent, descent and leading.

Syntax int32 CharHigh(void)

Return value Returns the character height (in pixels) for the current font.

Example This example draws an image that has three lines of text drawn in three rows.

```
void Test_CharHigh( TGC* gc )
{
  char *s[] =
  {
    "1st line",
    "2nd line",
    "3rd line"
  };
  INT32 i, x, y;

  //*** Setup look-up-table for colors white and black
  gc->SetLut( BW );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Select the BIG30 font via index.
  gc->SelectFont( "BIG30" );

  //*** Set the text color to be white
  gc->Transparent( TStdColor::white );

  x = 8;
  y = gc->CharHigh();
  for (i = 0; i <= 2; ++i)
  {
    gc->DrawString(x, y, s[i]);
    y += gc->CharHigh();
  }
}
```

CharWideImage

Class Font management

Description Calculates and returns the image width (in pixels) of the specified ASCII character. If a character code in the string is not defined in the font, then the width of the 'undefined character' is used for that character code.

Syntax int32 CharWideImage(int32 *code*)

code
ASCII character code.

Example This example draws an image that has a red diagonal line of text starting from the upper left corner of the screen. Each character should fit within a series of vertical white lines with no overlap.

```
void Test_CharWideImage( TGC* gc )
{
  char  c;
  INT32 w, h, x, x1, y, y2, i;
  char s[] = "QUANTUM DATA";

  //*** Setup look-up-table for colors white and black and Red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Select the BIG30 font via name
  gc->SelectFont( "BIG30" );

  //*** Initialize variables
  x  = gc->CharWideMax();
  y  = h = gc->CharHigh();
  y2 = h * strlen( s );
  i  = 0;

  //*** Draw the initial column line
  gc->Set_color1( TStdColor::white );
  gc->DrawLine( x - 1, 0, x - 1, y2 );

  while( (c = s[i])  != '\0' )
  {
    //*** Determine the current characters image width
    w = gc->CharWideImage(c);

    //*** Draw a white column line
    x1 = x + w + 1;
```

```
    gc->Set_color1( TStdColor::white );
    gc->DrawLine( x1, 0, x1, y2 );

    //*** Set the text color to be red
    gc->Transparent( TStdColor::red );

    //*** Draw a character
    gc->DrawChar( x, y, c );

    //*** Move to the next character column
    y += h;
    x += w + 2;
    i++;
  }
}
```

CharWideMax

Class Font management

Description Returns the width (in pixels) of the widest character in the current font.

The returned value is the sum of the character image width and the space preceding the next character to the right. The character image is the bitmap containing the character pattern.

Syntax int32 CharWideMax(void)

Return value Returns the width (in pixels) of the widest character in the current font. The returned value is the sum of the character image width and the space preceding the next character to the right.

Example This example demonstrates the CharWideMax() function.

```
void Test_CharWideMax( TGC* gc )
{
  char   c;
  INT32  w, h, x, y,i;
  char   s[] = "QUANTUM DATA";

  //*** Setup look-up-table for colors white and black
  gc->SetLut( BW );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Select the BIG30 font via name
  gc->SelectFont( "BIG30" );

  //*** Set the text color to be white
  gc->Transparent( TStdColor::white );

  //*** Get the text's dimensions
  x = w = gc->CharWideMax();
  y = h = gc->CharHigh();

  //*** Draw the text one char down and one char over from the last
  i = 0;
  while( (c = s[i])  != '\0' )
  {
    gc->DrawChar( x, y, c );

    y += h;
    x += w;
    i++;
  }
}
```

CloseVuport

Class Windows

Description Closes the viewport identified by the specified index.

The viewport must be open at call. If designated viewport is currently selected, viewport 0 is selected in its place. Viewport 0 can never be closed.

Syntax int32 CloseVuport(int32 *index*)

index
Index of the viewport.

Return value When the function is called with a valid index, the value 0 is returned to confirm that the viewport was closed as requested. When the function is called with an invalid index, a value of -1 is returned to indicate that no action was taken.

Example This example draws an image that has two pattern filled rectangles.

```
int vindex;
      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      SelectPatn(CHECKER8);
      /* Viewport 0's color0 = red and color1 = blue */
      Opaque(blue,red);
      vindex = OpenVuport();  /* Open vuport 1 */

      /* Viewport 1 inherits color0 from viewport 0 */
      SetColor1(green);   /* Viewport 1's color1 = green not blue */

      /* Fill square with blue-and-green pattern */
      PatnFillRect(96, 96, 272, 192);

      /* green is not color1 anymore, it is blue */
      CloseVuport(vindex);
      PatnFillRect(96, 96, 272, 192);
```

ColorFillVuport

Class Screen fills

Description Sets the entire extent of the active portion of memory to specified color.

Syntax ColorFillVuport(int32 *color*)

color
Desired color.

Example This example draws an image with a red background.

```
void Test_ColorFillVuport( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Fill the screen with red
  gc->ColorFillVuport( TStdColor::red );
}
```

CopyMatrix

Class 3-D methods

Description Copies a 4x4 input matrix to a 4x4 output matrix.

Both the input matrix *matrix_in* and the output matrix *matrix_out* are stored in 16-element arrays of type FIX.

Syntax CopyMatrix(int32 *matrix_in*[ ], int32 *matrix_out*[ ])

matrix_in[]
Input matrix.

matrix_out[]
Output matrix.

Example This example demonstrates the CopyMatrix() function. (This code produces no image.)

```
typedef long FIX;
      {
       FIX matrixin[16];
       FIX matrixout[16];
                .
                .
                .
       CopyMatrix(matrixin, mmatrixout);
      }
```

CopyVertex

Class 3-D methods

Description Copies an input vertex list to an output vertex list.

A vertex is stored as three consecutive 32-bit coordinate values X, Y, and Z. Each array contains 3n 32-bit elements.

Syntax CopyVertex(int32 *n*, int32 *vertex_in*[ ], int32 *vertex_out*[ ])

n
The number of vertices that are copied from the input vertex list to the output vertex list.

vertex_in[]
Input matrix consisting of an array of 32-bit fixed-point values.

vertex_out[]
Output matrix consisting of an array of 32-bit fixed-point values.

Example This example demonstrates the CopyVertex() function. (This code produces no image.)

```
typedef long FIX;

      /* Copy 5 vertices from vertex_in[] to vertex_out[] */
      /* Each vertex consumes 3 storage elements in an  */
      /* array, and the minimum array size is 3*5 = 15  */
      FIX vertexin[3*5], vertexout[3*5];

      CopyVertex(5, vertexin, vertexout);
```

CopyVuport

Class Windows

Description Copies all attributes of the source viewport to the destination viewport.

Both the source and the destination viewports must be opened before calling the CopyVuport() function. The destination viewport automatically becomes the active viewport.

Syntax int32 CopyVuport(int32 *index1*, int32 *index2*)

index1

Source viewport corresponding to the value returned by the OpenVuport() function when the viewport was opened.

index2

Destination viewport.

Return value If neither viewport was previously opened, a value of -1 is returned to indicate that an error was detected and that no viewport was copied. Otherwise, a value of 0 is returned to indicate that the viewport was successfully copied.

Example This example draws an image that has 3 solid filled rectangles.

```
/* Create 2 new viewports identical to the first,
      but located in different areas of the screen   */
      int index[4];    /* viewport indices */

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      /* Open viewport 1 */
      index[1] = OpenVuport();
      SizeVuport(150,300);
      MoveVuport(10,50);
      SelectVuport(index[1]);
      Transparent(white);
      FillRect(1000, 1000, -10, -10);
```

```
/* Make two new viewports similar to first */
/* create viewport 2 */
index[1] = OpenVuport();
/* create viewport 3 */
index[2] = OpenVuport();
CopyVuport(index[1],index[2]);
CopyVupor(index[1],index[3]);

/* Move viewport 3 to right of viewport 1 */
MoveVuport(340,50);
Transparent(red);
FillRect(1000, 1000, -10, -10);

/* Move viewport 2 between other two viewports */
SelectVuport(index[2]);
MoveVuport(170,50);
Transparent(blue);
FillRect(1000, 1000, -10, -10);
```

CurrentVuport

Class Windows

Description Returns the index of the current viewport as an int32.

Syntax int32 CurrentVuport(void)

Return value Returns the index of the current viewport as an int32.

Example This example draws an image that has a pattern filled oval and a pattern filled rectangle.

```
int index, v;
      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      SetClipRect(320, 480, 0, 0);


      /* Open viewport 1 and change its colors and pattern */
      v = OpenVuport();
      index = CurrentVuport();
      MoveVuport(320, 0);
      Opaque(blue,green);
      SelectPatn(CHECKER8);

      /* Display viewport 0's colors and pattern */
      SelectVuport(0);
      Opaque(red,white);
      SelectPatn(CHECKER8);

      PatnFillOval(300, 460, 10, 10);

      /* Display viewport 1's colors and pattern */
      SelectVuport(v);
      PatnFillRect(300, 460, 10, 10);
```

CurtainFillRect

Class Filling primitives with solid

Description Fills a specified rectangular area by copying either the top row or left column into the balance of the rectangle.

Syntax CurtainFillRect(int32 *width*, int32 *height*, int32 *xleft*, int32 *ytop*, int32 *mode*)

width
Width of rectangle.

height
Height of rectangle.

xleft
Coordinate of left edge of rectangle.

ytop
Coordinate of top edge of rectangle.

mode
Area to be copied (left-column or top-row).

Example This example draws an image that has a rectangle of blue horizontal lines and a rectangle of vertical red, green, blue lines.

```
void Test_CurtainFillRect( TGC* gc )
{
  INT32 x, y;

  //*** Setup look-up-table for colors white and black and Red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the text color to be blue
  gc->Transparent( TStdColor::blue );

  //*** Draw some vertical lines along the right edge of the screen
  for( x = 0, y = 0; y < 100; y += 5 )
  {
    gc->DrawLine( x, y, x, y+2 );
  }

  //*** Curtain Fill the vertical lines, filling left to right
  gc->CurtainFillRect( 100, 100, 0, 0, 1 );

  //*** Draw some horizontal lines along the top edge
```

```
  for( x = 150, y = 0; x < 255; x += 15 )
  {
    gc->Set_color1( TStdColor::red );
    gc->DrawLine( x, y, x+5, y );
    gc->Set_color1( TStdColor::green );
    gc->DrawLine( x+6, y, x+10, y );
    gc->Set_color1( TStdColor::blue );
    gc->DrawLine( x+11, y, x+14, y );
  }

  //*** Curtain fill the horizontal lines, top to bottom
  gc->CurtainFillRect( 106, 200, 150, 0, 0 );
}
```

DottedOval

Class Drawing primitives

Description Draws the dotted outline of an ellipse given the minimum enclosing rectangle in which the ellipse is inscribed and the spacing angle between the dots. The ellipse is in standard position, with its major and minor axes parallel to the coordinate axes. The dotted outline is one pixel thick, and is drawn in the current COLOR1.

Syntax DottedOval(int32 *width*, int32 *height*, int32 *xleft*, int32 *ytop*, int32 *step*)

w
Width of the enclosing rectangle.

h
Height of the enclosing rectangle.

xleft, *ytop*
Coordinates of the top left corner of the rectangle.

step
Dot spacing (degrees).

Example This example draws an image that has a white dashed oval and a green dashed oval.

```
/* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      Transparent(white);
      DottedOval( 300, 200, 170, 140, 5);
      Transparent(green);
      DottedOval( 200, 100, 220, 190, 3);
```

DrawCenterMark

Class Drawing composites

Description Draws a cross in the center of the screen. The length of each line in the cross is 5% of the width or height. The cross is drawn in the current COLOR1.

Syntax DrawCenterMark(void)

Example This example draws an image that has a white crosshair in the center of the screen.

```
void Test_DrawCenterMark( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the color to be white
  gc->Transparent( TStdColor::white );

  //*** Draw a cross in the center of the screen
  gc->DrawCenterMark();
}
```

DrawChar

Class Text

Description Draws a single bitmapped character to the specified screen coordinates. The character is drawn using the current font.

Syntax int32 DrawChar(int32 *x*, int32 *y*, int32 *code*)

x
Coordinate of the horizontal position at left edge of character.

y
Coordinate of the vertical position at the baseline of the string (not at the top of the string).

code
Pointer to a character.

Return value The return value is the x coordinate of the next character position to the right of the specified character. The x value is given in viewport-relative coordinates. If the character lies entirely above or below the window, the unmodified starting x coordinate is returned.

Example This example draws an image that has white letters arranged in a circle about the middle of the screen.

```
void Test_DrawChar( TGC* gc )
{
  char  c;
  INT32 x, y;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
   gc->BlackFillVuport();

  //*** Select the OPIX9 font via name
  gc->SelectFont( "OPIX9" );

  //*** Set the text color to be white
  gc->Transparent( TStdColor::white );

  x = 0;
  y = -170 << 16;

  //*** Draw the letters 'A' through 'Z'
  for( c = 'A'; c < 'Z' ; ++c )
  {
```

```
    //*** draw the characters on screen
    gc->DrawChar( (x >> 16) + 304, (y >> 16) + 244, c );
    x += y >> 3;
    y -= x >> 3;

    //*** draw the characters on screen
    gc->DrawChar( (x >> 16) + 304, (y >> 16) + 244, c - 'A' + 'a' );
    x += y >> 3;
    y -= x >> 3;
  }
}
```

DrawCross

Class Drawing composites

Description Draws a cross in the center of the screen using the current COLOR1. The length of the lines are the entire screen size.

Syntax DrawCross(void)

Example This example draws an image that has a white cross on the screen, dividing it into 4 parts.

```
void Test_DrawCross( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the color to be white
  gc->Transparent( TStdColor::white );

  //*** Draw a cross in the center of the screen
  gc->DrawCross();
}
```

DrawFormat

Class Drawing composites

Description Fills the fixed rectangle using COLOR0 and draws the outline of the rectangle along with the information about the used format using COLOR1.

Syntax DrawFormat(int32 *xleft*, int32 *ytop*)

xleft, ytop
Coordinates of the top left corner of the rectangle.

Example This example draws an image that has two FORMAT information boxes.

```
void Test_DrawFormat( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->Set_color1( TStdColor::green );
  gc->Set_color0( TStdColor::red   );

  gc->SelectFont( "OPIX9" );

  gc->DrawFormat( 320, 200 );
  gc->DrawFormat( 320, 280, TStdColor::white );
}
```

DrawGrid

Class Drawing composites

Description Draws grids of different types.

Syntax DrawGrid(int32 *xboxes*, int32 *yboxes*, int32 *version*)

xboxes
Number of boxes along the horizontal direction.

yboxes
Number of boxes along the vertical direction.

version
Grid version, where:

1= hatch

2= dot

3= dot hatch

4= outline

5= outline dot

6= outline hatch

7= outline dot hatch

13= triple outline hatch

15= triple outline dot hatch

21=fractional outline hatch

23=fractional outline dot hatch

29=fractional triple outline hatch

31=fractional triple outline dot hatch

Example This example draws an image that has a dotted hatch grid 16*12.

```
void Test_DrawGrid( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->Transparent( TStdColor::green );
  gc->DrawGrid( 16, 12, 31 );
}
```

DrawGrill

Class Drawing composites

Description Draws a grill in a specified window.

Syntax DrawGrill(int32 *width*, int32 *height*, int32 *x*, int32 *y*, int32 *onwidth*, int32 *offwidth*, int32 *axis*)

width
Width of rectangular area in which grill is drawn.

height
Height of rectangular area in which grill is drawn.

x, y
Coordinates for the location of the top left corner of rectangular area.

onwidth
Width of the filled bars.

offwidth
Width of the unfilled bars.

axis
0 = Horizontal bars

1 = Vertical bars

2 = Both horizontal and vertical bars

Example This example draws an image that has vertical and horizontal green bars.

```
void Test_DrawGrill( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->Transparent( TStdColor::green );
  gc->DrawGrill( 600, 300, 30, 30, 20, 15, 2 );
}
```

DrawLimits

Class Drawing composites

Description Draws screen limits using the current COLOR1. It draws corner markers, centermarker, and markers at the center of the screen sides.

Syntax DrawLimits(void)

Example This example draws an image that has white corner markers, centermarker, and markers at the center of the screen sides.

```
void Test_DrawLimits( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the color to be white
  gc->Transparent( TStdColor::white );

  //*** Draw a cross with the other limits on the screen
  gc->DrawLimits();
}
```

DrawLine

Class Drawing primitives

Description Uses Bresenham's algorithm to draw a line from the starting point to the ending point. Arguments x1 and y1 specify the starting coordinates. Arguments x2 and y2 specify the ending coordinates. The line is one pixel in thickness and is drawn in the current COLOR1.

Syntax DrawLine(int32 *x1*, int32 *y1*, int32 *x2*, int32 *y2*)

x1, y1
Starting coordinates.

x2, y2
Ending coordinates.

Example This example draws an image that has many lines drawn around a circle centered in the middle of the screen.

```
void Test_DrawLine( TGC* gc )
{
  INT32 i, x1 = -40, y1 = 80, x2 = 80, y2 = 280;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut(BW|HUERGB);

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Draw 202 lines in different orientations and colors
  for( i = 202; i > 0; --i )
  {
    // Alternate the line colors (rgb)
    switch( i % 3 )
    {
      case 0:
        gc->Set_color1( TStdColor::red );
        break;

      case 1:
        gc->Set_color1( TStdColor::green );
        break;

      case 2:
        gc->Set_color1( TStdColor::blue );
    }

    // Draw a line
    gc->DrawLine( x1 + 305, y1 + 222, x2 + 305, y2 + 222 );
```

```
    // Move to the next line
    x1 += y1 >> 5;
    y1 -= x1 >> 5;
    x2 += y2 >> 5;
    y2 -= x2 >> 5;
  }
}
```

DrawOval

Class Drawing primitives

Description Draws the outline of an ellipse given the minimum enclosing rectangle in which the ellipse is inscribed. The ellipse is in standard position, with its major and minor axes parallel to the coordinate axes. The outline is one pixel thick, and is drawn in the current COLOR1.

Syntax DrawOval(int32 *width*, int32 *height*, int32 *xleft*, int32 *ytop*)

width
Width of the enclosing rectangle.

height
Height of the enclosing rectangle.

xleft, ytop
Coordinates of the top left corner of the enclosing rectangle.

Example This example draws an image that has white ovals of various sizes.

```
void Test_DrawOval( TGC* gc )
{
  INT32 w, h, x, y;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the color to be white
  gc->Transparent( TStdColor::white );

  //*** Draw ellipses of various sizes
  for( w = 0, x = 4; w < 33; ++w, x += w + 3 )
    for( h = 0, y = 4; h < 28; ++h, y +=h + 3 )
      gc->DrawOval( w, h, x, y );
}
```

DrawPoint

Class Drawing primitives

Description Draws a single pixel at the specified position. The pixel is drawn using the current COLOR1.

Syntax DrawPoint(int32 *x*, int32 *y*)

x, y
Coordinates of the designated pixel.

Example This example draws an image that has figure 8 lines in green.

```
void Test_DrawPoint( TGC* gc )
{
  INT32 i, x, y, xy, yx;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** set the color to be green
  gc->Transparent( TStdColor::green );

  x = xy = 0;
  y = yx = 200;
  for( i = 1200; i > 0; --i )
  {
    gc->DrawPoint( x + 320, y + 240);
    x  += yx >> 4;
    yx -= x  >> 4;
    y  += xy >> 5;
    xy -= y  >> 5;
  }
}
```

DrawPolyLine

Class Drawing primitives

Description Draws multiple lines. For example, the first line to be drawn is specified in the first two elements, linelist [0] and linelist [1]. Assume that these contain index values 4 and 7, respectively. The starting coordinates for the line are contained in ptlist [2*4] and ptlist [2*4+1]. The ending coordinates are contained in ptlist [2*7] and ptlist [2*7+1].

The individual elements of the linelist array are assigned as follows.

- linelist[0] = starting point of line 0
- linelist[1] = ending point of line 0
- linelist[2] = starting point of line 1
- linelist[3] = ending point of line 1
- linelist[2n] = starting point of line n-1
- linelist[2n+1] = ending point of line n-1

The individual elements of the ptlist array are assigned as follows:

- ptlist[0] = x coordinate value for point 0
- ptlist[1] = y coordinate value for point 0
- ptlist[2] = x coordinate value for point 1
- ptlist[3] = y coordinate value for point 1
- ptlist[2m] = x coordinate value for point m-1

Syntax DrawPolyline(int32 *n*, TQD_Point *linelist*[ ], TQD_Point *ptlist*[ ])

n
Number of lines to be drawn.

linelist[]
List of lines in an array of type short. Each element in the *linelist* array is an index into the *ptlist* array. Each pair of adjacent 16-bit elements in the *linelist* array is a pair of indices into the *ptlist* array.

ptlist[]
List of points. The *ptlist* array contains the XY coordinates of the starting and ending points for each line. Each pair of adjacent 16-bit elements in the *ptlist* array is an X coordinate followed by a Y coordinate.

Example This example draws an image that has a white cube.

```
void Test_DrawPolyLine( TGC* gc )
{
```

```
  POINT ptlist[7];
  POINT linelist[9];

  ptlist[0] = 0x017C00C8;
  ptlist[1] = 0x01E000C8;
  ptlist[2] = 0x01E0012C;
  ptlist[3] = 0x017C012C;
  ptlist[4] = 0x0154010E;
  ptlist[5] = 0x015400AA;
  ptlist[6] = 0x01BB00AA;


  linelist[0] = 0x00000001;
  linelist[1] = 0x00010002;
  linelist[2] = 0x00020003;
  linelist[3] = 0x00030004;
  linelist[4] = 0x00040005;
  linelist[5] = 0x00050006;
  linelist[6] = 0x00060001;
  linelist[7] = 0x00030000;
  linelist[8] = 0x00050000;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the color to be white
  gc->Transparent( TStdColor::white );

  //*** Draw a cube
  gc->DrawPolyLine( 9, linelist, ptlist );
}
```

DrawRect

Class Drawing primitives

Description Draws the outline of a rectangle.

Syntax DrawRect(int32 *width*, int32 *height*, int32 *xleft*, int32 *ytop*)

width
Width of the rectangle.

heigth
Height of the rectangle.

xleft, *ytop*
Coordinates of the top left corner of the rectangle.

Example This example draws an image that has white rectangles drawn one inside another.

```
void Test_DrawRect( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** set the color to be white
  gc->Transparent( TStdColor::white );

  gc->DrawRect( 440, 280, 100, 100 );
  gc->DrawRect( 420, 30,  110, 110 );
  gc->DrawRect( 220, 220, 110, 150 );
  gc->DrawRect( 190, 150, 340, 150 );
  gc->DrawRect( 190, 60,  340, 310 );
}
```

DrawStep

Class Drawing composites

Description Fills the fixed rectangle using COLOR0 and draws the outline of the rectangle along with the step number of the image in the sequence using COLOR1. If there is not a sequence running, display 'none'.

Syntax DrawStep(int32 *x*, int32 *y*)

x, y
Coordinates of the center of the rectangle.

Example This example draws an image that has a box with the sequence step being drawn (none).

```
/* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      SetColor1(white);
      SetColor0(blue);

      SelectFont(OPIX9);
      /* (x, y) is the center of the box */
      DrawStep(320L,240L);
```

DrawString

Class Text

Description Draws a string of characters to the specified position on the screen. The string is drawn using the current font.

Syntax int32 DrawString(int32 *x*, int32 *y*, char* *s*)

x
Horizontal position at the left edge of the string.

y
Vertical position at the baseline of the string (not the top of the string)

s
Character string in a standard C format: a sequence of 8-bit ASCII character codes terminated by a NULL character (ASCII code = 0).

Return value The return value is the x coordinate of the next character position to the right of the string. The x value is given in viewport-relative coordinates. If the string lies entirely above or below the window, the unmodified starting x coordinate is returned.

Example This example draws an image that has “hello world” drawn 50 times in a circular pattern.

```
void Test_DrawString( TGC* gc )
{
  INT32  i;
  INT32  x, y;
  char   *s;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->Transparent( TStdColor::white );

  gc->SelectFont( "OPIX9" );
  s = "Hello world.";
  x = 0;
  y = 200 << 16;

  //*** Write 'Hello world' to the screen 50 times
  for( i = 50; i> 0; --i )
  {
    gc->DrawString( (x >> 16) + 272, (y >> 16) + 244, s );
    x += y >> 3;
    y -= x >> 3;
  }
}
```

DrawTBox

Class Drawing composites

Description Fills the fixed rectangle using COLOR0 and draws the outline of the rectangle along with the information using COLOR1. This rectangle shows the information about the current image displayed, format used, horizontal and vertical rates.

Syntax DrawTBox(int32 *x*, int32 *y*)

x, y
Coordinates of the center of the rectangle.

Example This example draws an image that has two information boxes.

```
void Test_DrawTBox( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->Set_color1( TStdColor::green );
  gc->Set_color0( TStdColor::red   );

  //*** Show the information about the current image displayed in a box
  gc->DrawTextBox( 320, 200 );
  gc->DrawTextBox( 320, 280, TStdColor::white );
}
```

DrawText

Class Text

Description

Syntax

Return value

Example This example draws an image that has “hello world” drawn 50 times in a circular pattern.

```
void Test_DrawText( TGC* gc )
{
  INT32  i;
  INT32  x, y;
  char   *s;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->Transparent( TStdColor::white );

  gc->SelectFont( "OPIX9" );
  s = "Hello world.";
  x = 0;
  y = 200 << 16;

  //*** Write 'Hello world' to the screen 50 times
  for (i = 50; i> 0; --i)
  {
    gc->DrawText( (x >> 16) + 272, (y >> 16) + 244, s );
    x += y >> 3;
    y -= x >> 3;
  }
}
```

DrawTriangle

Class Drawing primitives

Description Draws the outline of a triangle. This triangle is described by the coordinates of its three vertices). The outline is one pixel in thickness, and is drawn in the current COLOR1. The DrawTriangle() function uses the DrawPolyLine() function to connect the three vertices.

Syntax DrawTriangle(int32 *x0*, int32 *y0*, int32 *x1*, int32 *y1*, int32 *x2*, int32 *y2*)

x0, y0
Coordinates of vertex 0.

x1, y1
Coordinates of vertex 1.

x2, y2
Coordinates of vertex 2.

Example This example draws an image that has 3 triangles: a blue triangle and a green triangle inside of a white triangle.

```
void Test_DrawTriangle( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the color to be white
  gc->Transparent( TStdColor::white );

  gc->DrawTriangle( 100, 100, 500, 100, 320, 400 );

  //*** Set the color to be blue
  gc->Transparent( TStdColor::blue );
  gc->DrawTriangle( 160, 160, 420, 160, 320, 320 );

  //*** Set the color to be green
  gc->Transparent( TStdColor::green );
  gc->DrawTriangle( 240, 220, 340, 220, 320, 260 );
}
```

EncloseFillRect

Class Filling primitives with solid

Description Draws a filled-in rectangle. The dimensions of the rectangle are automatically computed to completely cover all of the pixels in the area whose boundaries are defined by the points in a point list.

Syntax EncloseFillRect(int32 n, TQD_Point ptlist[ ])

n
Number of points in *ptlist*.

ptlist[]
Pointer to point list.

Example This example draws an image that has a red filled rectangle near the upper center of the screen.

```
static short xy[] = {
         20,20, 60,40, 120,20, 300,100
      };

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      Transparent(red);
      EncloseFillRect(4, xy);
```

FillOval

Class Filling primitives with solid

Description Draws an ellipse that is solid-filled with the current COLOR1. The ellipse is in standard position, with its major and minor axes parallel to the coordinate axes. The ellipse is defined by the minimum enclosing rectangle in which it is inscribed.

Syntax FillOval(int32 width, int32 height, int32 xleft, int32 ytop)

width
Width of the enclosing rectangle.

height
Height of the enclosing rectangle.

xleft, ytop
Coordinates of the top left corner of the enclosing rectangle.

Example This example draws an image that has green filled elipses in different sizes.

```
void Test_FillOval( TGC* gc )
{
  INT32 w, h, x, y;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** set the color to be green
  gc->TranspOFF();
  gc->Set_color1( TStdColor::green );

  //*** Fill ellipses of various sizes
  for( w = 0, x = 4; w < 33; ++w, x += w + 3 )
    for( h = 0, y = 4; h < 28; ++h, y += h + 3 )
      gc->FillOval( w, h, x, y );
}
```

FillPolygon

Class Filling primitives with solid

Description Fills a polygon given a list of points representing the vertices of the polygon. No restrictions are placed on the shape of the polygon filled by the function: edges can cross each other, filled areas can contain holes. The polygon is solid-filled with COLOR1.

Syntax FillPolygon(int32 n, TQD_Point ptlist[ ])

n
Number of vertices in the polygon.

ptlist[]
An array of type IntPoint2, which is a structure of the short x and y coordinates of the vertices. The coordinates of the vertices are at (point[i].x, point[i].y) for i in [0..nvert-1].

Example This example draws an image that has a star-shaped filled polygon.

```
void Test_FillPolygon( TGC* gc )
{
  INT32 n = 5;
  POINT points[5];

  points[0] = 0x003200F0;
  points[1] = 0x026C0082;
  points[2] = 0x00B401C2;
  points[3] = 0x01400032;
  points[4] = 0x01A401A4;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();
  gc->TranspOFF();

  //*** Set the color to be green
  gc->Set_color1( TStdColor::green );
  gc->FillPolygon( n, points );
}
```

FillRect

Class Filling primitives with solid

Description Draws a rectangle that is solid-filled with the current COLOR1.

Syntax FillRect(int32 width, int32 height, int32 xleft, int32 ytop)

width
Width of the rectangle.

height
Height of the rectangle.

xleft, ytop
Coordinates of the top left corner of the rectangle.

Example This example draws an image that has 5 filled rectangles of various sizes and colors.

```
void Test_FillRect( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the color to be white
  gc->TranspOFF();
  gc->Set_color1( TStdColor::white );

  //*** Draw a filled rectangle
  gc->FillRect( 440, 280, 100, 100 );

  gc->Set_color1( TStdColor::red );
  gc->FillRect( 420, 30, 110, 110 );

  gc->Set_color1( TStdColor::green );
  gc->FillRect( 220, 220, 110, 150 );

  gc->Set_color1( TStdColor::blue );
  gc->FillRect( 190, 150, 340, 150 );

  gc->Set_color1( TStdColor::black );
  gc->FillRect( 190, 60, 340, 310 );
}
```

FillTriangle

Class Filling primitives with solid

Description Draws a triangle that is solid-filled with the current COLOR1. This triangle is described by the coordinates of its three vertices.

Syntax FillTriangle(int32 x0, int32 y0, int32 x1, int32 y1, int32 x2, int32 y2)

x0, y0
Coordinates of vertex 0.

x1, y1
Coordinates of vertex 1.

x2, y2
Coordinates of vertex 2.

Example This example draws an image that has 3 multicolored, filled triangles.

```
void Test_FillTriangle( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();
  gc->TranspOFF();

  //*** Set the color to be white
  gc->Set_color1( TStdColor::white );
  gc->FillTriangle( 100, 100, 500, 100, 320, 400 );

  gc->Set_color1( TStdColor::blue );
  gc->FillTriangle( 160, 160, 420, 160, 320, 320 );

  gc->Set_color1( TStdColor::green );
  gc->FillTriangle( 240, 220, 340, 220, 320, 260 );
}
```

FixToInt16

Class Fixed point functions

Description Converts an array of fixed-pont numbers to short integers. Elements of input array are 32-bit two's complement fixed-point numbers whose 16 LSBs are to the right of the binary point. The conversion from fixed-point format is done by shifting the elements right by 16 (truncation).

Syntax short* FixToInt16(int n, long in_array[ ], short out_array[ ])

n
Number of elements to be converted.

in_array
Pointer to input array (elements are integers).

out_array
Pointer to output array (elements are fixed point).

Return value Returns a pointer to the first element of the output array.

Example This example draws an image that has a rotating triangle in the x-y plane.

```
typedef long FIX;
      static short ptlist[2][6] = { 0,-200, 30,15, -30,15 };
      static short connect[] = { 0, 1, 2 };
      FIX xy[6];
      unsigned short Which = 0;
      unsigned long frame = wait_frame(0)+1;
      int i;
      int ViewA, ViewB;

      /* If changing the origin it is better to use a temporary
      viewport */
      ViewA = CurrentVuport();
      ViewB = OpenVuport();
      SelectVuport(ViewB);

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
```

```
BlackFillVuport();

SetOrigin(320, 240);

Int16ToFix(6, ptlist[Which], xy);

/* Draw a rotating triangle in the x-y plane */
for (;;) {
      if(kbhit())break;
        for (i = 0; i <= 2; ++i) {
         xy[2*i] -= xy[2*i+1] >> 6L;
         xy[2*i+1] += xy[2*i] >> 6L;
      }
      FixToInt16(6, xy, ptlist[Which]);
      Which = Which ^ 1;
      WaitFrame(frame);
      frame+=1;
      Opaque(black,black);
      EncloseFillRect(4, ptlist[Which]);
      Transparent(red);
      Which = Which ^ 1;
      FillConvex(3, connect, ptlist[Which]);
      Which = Which ^ 1;
}
SelectVuport(ViewA);
CloseVuport(ViewB);
```

FixToInt32

Class Fixed point functions

Description Converts an array of fixed-point numbers to long integers. Elements of input array are 32-bit two's complement fixed-point numbers consisting of 16 bits to the left of binary point, and 16 bits to the right of binary point. Elements of the output array are 32-bit, two's complement integers (C type long). The conversion from fixed-point format is done by shifting the elements right by 16 (truncation with sign extension).

Syntax short* FixToInt32(int n, long in_array[ ], short out_array[ ])

n
Number of elements to be converted.

in_array
Pointer to input array (elements are integers).

out_array
Pointer to output array (elements are fixed point).

Return value Returns a pointer to the first element of the output array.

Example This example draws an image that is a rotating line.

```
 typedef long FIX;
static FIX xy1[2] = {0, 0xff6a0000 };
static long xy2[2][2];
unsigned short Which = 0;
unsigned long frame = wait_frame(0)+1;
int ViewA, ViewB;

/* If changing the origin it is better to use a temporary
viewport */
ViewA = CurrentVuport();
ViewB = OpenVuport();
SelectVuport(ViewB);

/* Establish number of image versions. */
SetVersions(0);

/* Setup look-up-table for colors white and black and colors red,
green and blue */
SetLut(BW|HUERGB);

init_canvas();

/* Clear the screen with black. */
BlackFillVuport();
```

```
SetOrigin(320, 240);

/* Draw a rotating line */
for(;;){
   if(kbhit())break;
   FixToInt32(2,xy1,xy2[Which]);
   Which = Which ^ 1;
   WaitFrame(frame);
   frame+=1;
   Opaque(black,black);
   DrawLine(0, 0, xy2[Which][0], xy2[Which][1]);
   Transparent(green);
   Which = Which ^ 1;
   DrawLine(0, 0, xy2[Which][0], xy2[Which][1]);
   Which = Which ^ 1;
   xy1[0] -= xy1[1] >> 6L;
   xy1[1] += xy1[0] >> 6L;
}
SelectVuport(ViewA);
CloseVuport(ViewB);
```

FrameOval

Class Filling primitives with solid

Description Solid-fills an ellipse-shaped frame with the current COLOR1. The frame consists of a filled region between two concentric ellipses. The portion of the screen enclosed is not altered. The outer oval is described by the minimum enclosing rectangle in which it can be circumscribed. The rectangle is specified in terms of its width, height, and the coordinates at its top left corner. The thickness of the frame in the x and y dimensions is given by two additional arguments, dx and dy, which specify the horizontal and vertical distances between the outer and inner ellipses.

Syntax FrameOval(int32 width, int32 height, int32 xleft, int32 ytop, int32 dx, int32 dy)

width
Width of the enclosing rectangle.

height
Height of the enclosing rectangle.

xleft, ytop
Coordinates of the top left corner of the enclosing rectangle.

dx
Horizontal distance between outer and inner ellipses.

dy
Vertical distance between the outer and inner ellipses.

Example This example demonstrates the FrameOval() function.

```
void Test_FrameOval( TGC* gc )
{
  INT32 i;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();
  gc->TranspOFF();

  for( i = 0; i <= 10; i++ )
  {
    gc->Set_color1( gc->RepPixel( i+1 ) );
    gc->FrameOval( 640 - i*60, 20 + i*46, i*30, 230 - i*23, 8, 8 );
  }
}
```

FrameRect

Class Filling primitives with solid

Description Solid-fills a rectangular frame with the current COLOR1. The frame consists of a filled region between two concentric rectangles. The portion of the screen enclosed is not altered.

Syntax FrameRect(int32 width, int32 height, int32 xleft, int32 ytop, int32 dx, int32 dy)

width
Width of the rectangle.

height
Height of the rectangle.

xleft, ytop
Coordinates of the top left corner of the rectangle.

dx
Horizontal distance between outer and inner rectangle.

dy
Vertical distance between the outer and inner rectangle.

Example This example draws an image that has a solid red framed rectangle.

```
void Test_FrameRect( TGC* gc )
{
  INT32 w, h, x, y, dx, dy;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();
  gc->TranspOFF();
  gc->Set_color1( TStdColor::red );

  w  = 480;
  h  = 360;
  x  =  80;
  y  =  60;
  dx =  40;
  dy =  30;

  gc->FrameRect( w, h, x, y, dx, dy );
}
```

FTransformX

Class 2-D methods

Description Transforms a normalized coordinate into an equivalent machine unit on x axis. xnormcoord is normalized to 1.0 for the minor axis.

Syntax double FTransformX(double xdoub)

xdoub
x coordinate to be transformed into machine units.

Return value This function returns double.

Example This example demonstrates the FTransformX function.

```
/* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      Transparent(white);
      ResetMatrix();
      /* draw a point at top of screen and the x coordinate in current
      normalized axis coordinates */
      DrawPoint( (int)FTransformX(0.75) ,0);
      /* draw a point at top of screen and where x=0.0 in current
      normalized axis coordinates.*/
      DrawPoint( (int)FTransformX(0.0) , 0);
```

FTransformY

Class 2-D methods

Description Transforms a normalized coordinate into an equivalent machine unit on y axis. The variable ynormcoord is normalized to 1.0 for the minor axis.

Syntax double FTransformY(double ydoub)

ydoub
y coordinate to be transformed into machine units.

Return value This function returns double.

Example This example demonstrates the FTransformY function.

```
/* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      Transparent(white);
      ResetMatrix();

      /* draw a point at left of screen and the y coordinate in current
      normalized axis coordinates */
      DrawPoint(0, (int)FTransformY(0.75)  );
      /* draw a point at top of screen and where y=0.0 in current
      normalized axis coordinates */
      DrawPoint(0, (int)FTransformY(0.0));
```

Get_Pmask

Class Register access

Description Returns the value of the plane mask (TMS34010 PMASK register). The plane mask designates which bits within a pixel are protected against writes, and affects all operations on pixels. The protected bits are replicated throughout the 32-bit plane mask in a manner similar to that used for the COLOR0 and COLOR1 values. The 1s in the plane mask specify protected bits in the destination pixel that cannot be modified, while the 0s specify bits that can be altered. The plane mask can be altered by means of a call to the Set_Pmask function.

Syntax uint32 Get_Pmask(void)

Example This example demonstrates the Get_Pmask() function. (This code produces no image.)

```
long mask;

      /* Establish number of image versions. */
      SetVersions(0);

      init_canvas();

      Set_Pmask(0x11111111);

      /* Write protect pixel bit #0 */
      mask = Get_Pmask();
      /* Return value = 0x00001111 */
```

Get_ppop

Class Register access

Description Returns the code for the current pixel processing operation (the PPOP field in the TMS34010's CONTROL register). The 5-bit PPOP code resides in the 5 LSBs of the return value; all higher order bits are 0s. The PPOP code determines the manner in which pixels are combined (logically or arithmetically) during drawing operations. A new PPOP code can be selected by means of the Set_ppop() function. Legal PPOP codes are in the range 0 to 21.

Syntax Get_ppop(void)

Return value Returns the code for the current pixel processing operation.

Example This example demonstrates the Get_ppop() function. (This code produces no image.)

```
long ppop;

      /* select XOR */
      Set_ppop(10);
      ppop = Get_ppop();
      /* returns PPOP = 10 */
```

GetAscent

Class Text

Description Returns the value of the ascent parameter for the current font. The ascent value is the number of vertical pixels from the baseline to the top of the tallest character in the font. The ascent for the current font is specified within the font structure; the GetAscent function retrieves the ascent value from the structure.

Syntax int32 GetAscent(void)

Return value Returns the value of the ascent parameter for the current font.

Example This example draws an image that has white text in a red box at the upper left corner of the screen.

```
void Test_GetAscent( TGC* gc )
{
  char *s = "Hello world.";
  INT32 w, h, x, y;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the font
  gc->SelectFont( "OPIX9" );

  //*** Draw text flush with top of screen
  x = y = 0;

  h = gc->GetAscent();
  w = gc->GetWidth(s);

  gc->Transparent( TStdColor::red );
  gc->FillRect( w, h, x, y );

  gc->Transparent( TStdColor::white );
  gc->DrawString( x, y+h, s );
}
```

GetColor0

Class Register access

Description Retrieves the 32-bit color value stored at LUT location 0.

Syntax uint32 GetColor0(void)

Return value Returns the color value as an uint32.

Example This example produces an image that has a pattern-filled rectangle of colors black and red.

```
unsigned long color0, color1;
char s[32];

/* Establish number of image versions. */
SetVersions(0);

/* Setup look-up-table for colors white and black and colors red,
 green and blue */
SetLut(BW|HUERGB);

init_canvas();

/* Clear the screen with black. */
BlackFillVuport();

SetColor0(black);
SetColor1(red);

color0=GetColor0();
color1=QD_GetColor1();

SelectFont(OPIX9);
itoaf(color1,s,16,0,"");
DrawText(100,100,s);

Opaque(color1,color0);
SelectPatn(CHECKER8);
PatnFillRect(140,140,240,150);
```

GetColor1

Class	Register access
Description	Retrieves the 32-bit color value stored at LUT location 1.
Syntax	uint32 QD_GetColor1(void)
Return value	Returns the color value as an uint32.
Example	See page 80.

GetDescent

Class Font management

Description Returns the value of the descent parameter for the current font. The descent is the vertical distance measured in pixels from the baseline to the lowest descender in the character set. The descent for the current font is specified within the font structure; the GetDescent function retrieves the descent value from the structure.

Syntax int32 GetDescent(void)

Return value Returns the value of the descent parameter for the current font.

Example This example draws an image that has white text with a red line drawn just below descenders.

```
void Test_GetDescent( TGC* gc )
{
  char  *s = "jumping jimminy";
  INT32 w, h, x, y;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->SelectFont( "OPIX9" );
  gc->Transparent( TStdColor::white );

  //*** Draw line just below descenders
  x = 0;
  y = gc->GetAscent();
  w = gc->GetWidth(s);
  h = gc->GetDescent();
  gc->DrawString( x, y, s );
  gc->Transparent( TStdColor::red );
  gc->FillRect( w, 1, x, y+h );
}
```

GetEdgeColor

Class Color

Description Sets and creates the edge color for each transition sample at the sine-square function.

Syntax int32 GetEdgeColor(int32 color, int32 sample)

color
Edge color name.

sample
Number of sine-squared transition samples.

Return value Returns the replicated color value of the sample.

Example This example demonstrates the GetEdgeColor function.

```
int w, i, j, e;
       unsigned long color[7][2] ;
       char s[32];
       char s1[] = "no of samples ";


      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for color black */
      SetLut(BLACK);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      /* Draw TV Bar100 (anti-aliasing effect) */
       color[0][0] = SetColorNamed("white");
       color[0][1] = SetEdgeColorNamed("white","yellow");
       color[1][0] = SetColorNamed("yellow");
       color[1][1] = SetEdgeColorNamed("yellow","cyan");
       color[2][0] = SetColorNamed("cyan");
       color[2][1] = SetEdgeColorNamed("cyan","green");
       color[3][0] = SetColorNamed("green");
       color[3][1] = SetEdgeColorNamed("green","magenta");
       color[4][0] = SetColorNamed("magenta");
       color[4][1] = SetEdgeColorNamed("magenta","red");
       color[5][0] = SetColorNamed("red");
       color[5][1] = SetEdgeColorNamed("red","blue");
       color[6][0] = SetColorNamed("blue");
       color[6][1] = SetEdgeColorNamed("blue","black");
```

```
/* Eighth bar is black. */

w=width/8;
e=GetEdgeColorMany();

if( w < (e + 1) ) return(TRUE);

for(i = 0;i < 7;i++)
{
  Transparent(color[i][0]);
  FillRect(w-e,1,(i*w),0);

  for(j = 0;j < e;j++)
  {
     Transparent(GetEdgeColor(color[i][1],j));
     DrawPoint(((i*w)+w-e+j),0);
  }
}

Opaque(black,black);
FillRect(w,1,(7*w),0);

/* Turn transparency off so black areas get filled too */
TranspOFF();
CurtainFillRect(width,height,0,0,0);

SelectFont(BIG30);
itoaf(e,s,10,0,"");
strcat(s1,s);
Transparent(color[0][0]);
DrawItalic(210,240,s1);
```

GetEdgeColorMany

Class Color

Description Returns the number of samples at the transition edge between colors.

Syntax GetEdgeColorMany(void)

Return value Returns the number of samples at the transition edge between colors.

Example This example demonstrates the GetEdgeColorMany function.

```
int w, i, j, e;
       unsigned long color[7][2] ;
       char s[32];
       char s1[] = "no of samples ";


      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for color black */
      SetLut(BLACK);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      /* Draw TV Bar100 (anti-aliasing effect) */
       color[0][0] = SetColorNamed("white");
       color[0][1] = SetEdgeColorNamed("white","yellow");
       color[1][0] = SetColorNamed("yellow");
       color[1][1] = SetEdgeColorNamed("yellow","cyan");
       color[2][0] = SetColorNamed("cyan");
       color[2][1] = SetEdgeColorNamed("cyan","green");
       color[3][0] = SetColorNamed("green");
       color[3][1] = SetEdgeColorNamed("green","magenta");
       color[4][0] = SetColorNamed("magenta");
       color[4][1] = SetEdgeColorNamed("magenta","red");
       color[5][0] = SetColorNamed("red");
       color[5][1] = SetEdgeColorNamed("red","blue");
       color[6][0] = SetColorNamed("blue");
       color[6][1] = SetEdgeColorNamed("blue","black");
       /* Eighth bar is black. */

       w=width/8;
       e=GetEdgeColorMany();
```

```
if( w < (e + 1) ) return(TRUE);

for(i = 0;i < 7;i++)
{
  Transparent(color[i][0]);
  FillRect(w-e,1,(i*w),0);

  for(j = 0;j < e;j++)
  {
     Transparent(GetEdgeColor(color[i][1],j));
     DrawPoint(((i*w)+w-e+j),0);
  }
}

Opaque(black,black);
FillRect(w,1,(7*w),0);

/* turn transparency off so black areas get filled too */
TranspOFF();
CurtainFillRect(width,height,0,0,0);

SelectFont(BIG30);
itoaf(e,s,10,0,"");
strcat(s1,s);
Transparent(color[0][0]);
DrawItalic(210,240,s1);
```

GetFirstCh

Class Font management

Description Returns the ASCII character code for the first character present in the current font. All characters whose codes are less than the return value correspond to "missing" characters; that is, their character images are omitted from the font structure.

Syntax int32 GetFirstChar(void)

Return value Returns the ASCII character code for the first character present in the current font.

Example This example draws an image that has 2 lines of white text with all visible ASCII characters displayed.

```
void Test_GetFirstCh( TGC* gc )
{
  UINT8 c;
  INT32 x, y;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Select the OPIX9 font via name
  gc->SelectFont( "OPIX9" );

  //*** set the text color to be white
  gc->Transparent( TStdColor::white );

  x = 10;
  y = 100;
  for( c = gc->GetFirstCh(); c <= gc->GetLastCh(); ++c, x +=
gc->CharWideMax() )
  {
    if( x > 600 )
    {
      x = 10;
      y += 50;
    }
    gc->DrawChar( x, y, c );
  }
}
```

GetFontMax

Class Font management

Description Returns the maximum number of fonts that can be installed simultaneously. If a value of n is returned, font indices can range from 0 to n-1.

Syntax int32 GetFontMax(void)

Return value Returns the maximum number of fonts that can be installed simultaneously. If a value of n is returned, font indices can range from 0 to n-1.

Example This example draws an image that has a numeric text which represents the maximum number of fonts available.

```
void Test_GetFontMax( TGC* gc )
{
  INT16 x, y, i;
  char  s1[32];

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** set the active color
  gc->Transparent( TStdColor::white );

  gc->SelectFont( "BIG30" );

  sprintf( s1, "%d", gc->GetFontMax() );
  gc->DrawString( 320, 240, s1 );
}
```

GetLastCh

Class Font management

Description Returns the ASCII character code for the last character present in the current font. All characters whose codes are greater than the return value correspond to "missing" characters; that is, their character images are omitted from the font structure.

Syntax int32 GetLastChar(void)

Return value Returns the ASCII character code for the last character present in the current font.

Example This example draws an image that has 2 lines of white text with all visible ASCII characters displayed.

```
void Test_GetLastCh( TGC* gc )
{
  UINT8 c;
  INT32 x, y;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Select the OPIX9 font via name
  gc->SelectFont( "OPIX9" );

  //*** Set the text color to be white
  gc->Transparent( TStdColor::white );

  x = 10;
  y = 100;
  for( c = gc->GetFirstCh(); c <= gc->GetLastCh(); ++c, x +=
gc->CharWideMax() )
  {
    if (x > 600)
     {
      x = 10;
      y += 50;
     }
     gc->DrawChar( x, y, c );
  }
}
```

GetLeading

Class Font management

Description Returns the leading value for the current font. The leading is the empty space between rows of text; that is, it is the number of vertical pixels from the lowest descenders of one row of text to the tallest characters of the line of text below it. This function retrieves the leading value from the font structure.

Syntax int32 GetLeading(void)

Return value Returns the leading value for the current font.

Example This example draws an image that has white text written above white lines, row by row, with increasing space from the left side of the screen.

```
void Test_GetLeading( TGC* gc )
{
  char  s[] = "Quantum Data...";
  INT32 x, y, dx;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the active color
  gc->Transparent( TStdColor::white );

  //*** Set the font
  gc->SelectFont( "OPIX9" );

  dx = (640 - gc->GetWidth(s)) / (480 / gc->CharHigh());
  for( x = 0, y = gc->GetAscent(); y < 480; x += dx,
  y += gc->CharHigh() )
  {
    gc->DrawString( x, y, s );
    gc->FillRect( 640, 1, 0, y + gc->GetDescent() +
    gc->GetLeading() /2 );
  }
}
```

GetLevelMax

Class Color

Description Gets the maximum luminance level. Level 0 is used to represent the minimum luminance level (black), while levelmax represents a maximum luminance level (red, green, yellow, etc.). Level max defaults to 255.

Syntax uint32 GetLevelMax(void)

Return value Returns the level as an uint32.

Example This example demonstrates the GetLevelMax function.

```
int i, x, w;
      char s[32];

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      w=width/16;
      x=(width-(w*16))/2;
      SetLevelMax(15);     /* level 0 -> level 15 */

      for(i=0;i<=GetLevelMax();i++)
      {
          Transparent(RepPixel(i));
          SelectColor(i);
          SetLevelRGB(0,i,0);
          FillRect(w,height,x,0);
          x=x+w;
       }

      itoaf(GetLevelMax(),s,10,0,"");
      SelectFont(OPIX9);
      /* Use max luminance (green) to write the text */
      Transparent(RepPixel(15));
      DrawString(320,240,s);
```

GetMoreSpace

Class Text

Description Gets the value of the count after drawing the text. Assume a text of five words with four spaces between its words. For n=0 means add nothing in between the words and the return value is 0. For n=1 means add one pixel in the first space (between the first and the second word) and the return value is 0. For n=4 means add one pixel in each space and return 0. For n=5 means just add one pixel between the words and the return value is 1.

Syntax GetMoreSpace(void)

Return value Returns amount of micro-spacing between some words.

Example This example draws an image that has a text in which space between some words is increasing.

```
void Test_GetMoreSpace( TGC* gc )
{
  INT16 x, y, i;
  char  *s;
  char  s1[32];
  INT32 os = gc->GetMoreSpace();

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the active color
  gc->Transparent( TStdColor::white );

  gc->SelectFont( "OPIX9" );

  s = "Note increasing spaces between words.";

  for( i = 0, y = 0; i < 20; ++i )
  {
    gc->AddMoreSpace(i);
    x = 320 - gc->GetWidth(s)/2;
    y += gc->CharHigh();
    gc->DrawText( x, y, s );
    sprintf( s1, "%d", gc->GetMoreSpace() );
    gc->DrawString( x - 40, y, s1 );
  }

  gc->AddMoreSpace( os );
}
```

GetPatnMax

Class Pattern management and primitives

Description Returns the maximum number of patterns that can be installed at any one time. The maximum number of patterns is a function of the size of the pattern table data structure.

Syntax int32 GetPatnMax(void)

Return value Returns the maximum number of patterns that can be installed at any one time. If return value is n, available range of vertices is 0 to n-1.

Example This example draws an image that has different pattern-filled rectangles.

```
void Test_GetPatnMax( TGC* gc )
{
  INT32 w, h, x, y, dx, dy, pmax, patn;

  //*** Setup look-up-table for colors white and black and colors reg,
green
  //    and blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen with black.
  gc->BlackFillVuport();

  gc->Set_color1( TStdColor::cyan  );
  gc->Set_color0( TStdColor::white );

  //*** Display all available patterns
  pmax = gc->GetPatnMax();
  x  = y = 0;
  w  = 84;
  h  = 68;
  dx = 96;
  dy = 80;
  gc->SelectPatn( 0 );
  for( patn = 0; patn < pmax; gc->SelectPatn( ++patn ) )
  {
    gc->PatnFillRect( w, h, x, y );
    if( (x += dx) > 640 - w )
    {
      x = 0;
      y += dy;
    }
  }
}
```

GetPixel

Class Bitmap icons

Description Returns the value of the pixel at coordinates (x,y). The coordinates are relative to the viewport origin. Given a pixel size of n bits, the pixel is contained in the n LSBs of the return value (the MSBs are 0s).

Syntax int32 GetPixel(int32 x, int32 y)

x, y
Coordinates of the pixel.

Example This example draws an image that has a solid filled rectangle with flipped text inside.

```
void Test_GetPixel( TGC* gc )
{
  INT32 xs, ys, xd, yd;
  char s[] = "topsy turvy";

  //*** Setup look-up-table for colors white and black and colors red,
green
  //*** and blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen with black.
  gc->BlackFillVuport();

  gc->Transparent( TStdColor::red );
  gc->FillRect   ( 150, 50, 0, 0 );

  gc->Transparent( TStdColor::white );
  gc->SelectFont ( "OPIX9" );
  gc->DrawString ( 0, gc->GetAscent(), s );

  //*** Flip and mirror original text
  for( ys = 0, yd = 29; ys <= 19; ++ys, --yd )
    for( xs = 0, xd = 89; xs <= 89; ++xs, --xd )
      gc->PutPixel( gc->GetPixel(xs,ys), xd, yd );
}
```

GetPsize

Class Register access

Description Returns the pixel depth currently being used by the generator.

Syntax int32 GetPSize(void)

Return value Returns the pixel depth (1, 2, 4, 8, 16, or 24 bits) currently being used by the generator as a int32.

Example This example demonstrates the GetPsize function.

```
long psize;
psize = GetPsize();
```

GetRect

Class Bitmap icons

Description Copies pixels contained in a rectangular area of the screen into a specified packed pixel array. The source rectangle is clipped to positive x and y coordinate space before being copied. Portions of the source array lying in negative x-y space are not copied, and the corresponding portions of the destination array remain unaltered. Note that portions of the source array lying outside the current viewport and clipping rectangle are NOT clipped unless they happen to lie in negative x-y space.

Syntax GetRect(int32 width, int32 height, int32 xsrc, int32 ysrc, int32 destarray[ ], int32 destpitch)

width
Width of the source rectangle.

height
Height of the source rectangle.

xsrc, ysrc
Coordinates of the top left corner of the source rectangle.

destarray[]
Pointer to the start of the destination array. The destination array must be large enough to contain the source array. The minimum number of bits required in the destination array is calculated as (height * pitch).

destpitch
Pointer to the pitch of the destination array. The array pitch is the difference in the starting addresses of two adjacent rows of the array. The pitch must be a positive multiple of the pixel size. The minimum pitch is (width * psize), where psize represents the pixel size.

Example This example draws an image that has a shape surrounded by copies of itself.

```
void Test_GetRect( TGC* gc )
{
  INT32   w, h, pitch32, i;
  INT32   x, y;
  INT32   bitwidth;
  UINT32 *buf;

  w = 80;
  h = 60;
  x = 280;
  y = 210;

  bitwidth = gc->Get_psize() * w;
```

```
  pitch32 = bitwidth / 32;
  if( bitwidth % 32 ) pitch32++;

  buf = (UINT32*)malloc( pitch32 * h * sizeof(UINT32) );
                                                  if( buf == NULL )return;

  //*** Setup look-up-table for colors white and black and colors red,
green
  //*** and blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen with black.
  gc->BlackFillVuport();

  gc->Transparent( TStdColor::red );
  gc->FillRect( w-2, h-2, x+1, y+1 );
  gc->Opaque( TStdColor::green, TStdColor::blue );
  gc->SelectPatn( "CHECKER8" );
  gc->PatnFrameOval( w-2, h-2, x+1, y+1, 20, 15 );

  gc->GetRect( w, h, x, y, buf, pitch32 );

  for( i = 25, x = 0, y = -150<<16; i > 0; --i )
  {
    x += y >> 2;
    y -= x >> 2;
    gc->PutRect( buf, pitch32, w, h, (x>>16)+280, (y>>16)+210 );
  }

  free( buf );
}
```

GetTextSpace

Class Text

Description Gets the horizontal spacing between characters. Associated with each text font is a default spacing between a character and the character to its right.

Syntax int32 GetTextSpace(void)

Return value Amount of spacing between characters. The returned value can be positive or negative.

Example This example draws an image that has a text in which space between the characters is increasing.

```
void Test_GetTextSpace( TGC* gc )
{
  INT16 x, y, i;
  char  *s;
  char  s1[32];
  INT32 os = gc->GetTextSpace();

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** set the active color
  gc->Transparent( TStdColor::white );

  gc->SelectFont( "OPIX9" );

  s = "Note increasing spaces";

  for( i = 0, y = 0; i < 20; ++i )
  {
    gc->AddTextSpace(i);
    x = 320 - gc->GetWidth(s)/2;
    y += gc->CharHigh();
    gc->DrawString( x, y, s );
    sprintf( s1, "%d", gc->GetTextSpace() );
    gc->DrawString( x - 40, y, s1 );
  }

  gc->AddTextSpace( os );
}
```

GetTransp

Class Register access

Description Returns the current state of the transparency enable bit in the generator's control register.

Syntax uint32 GetTransp(void)

Return value Returns a 0 if transparency is off and a 1 if transparency is on. The return results are formatted as an uint32.

Example This example demonstrates the GetTransp() function. (This code produces no image.)

```
int t;

      /* Establish number of image versions. */
      SetVersions(0);

      init_canvas();

      t = GetTransp();  /* Returns value = 0 */
      TranspOn();
      t = GetTransp();  /* Returns value = 1 */
```

GetVuportMax

Class Windows

Description Returns the maximum number of viewports that can be opened at any one time.

Syntax short GetVuportMax(void)

Return value Returns the maximum number of viewports that can be opened at any one time. If the return value is a number n, the range of indices for available viewports is 0 to n-1.

Example This example demonstrates the GetVuportMax() function. (This code produces no image.)

```
int  v;

      v = GetVuportMax();
```

GetWidth

Class Text

Description Computes and returns the width in pixels of the specified ASCII string if drawn using the currently selected text font. The width is given as the number of pixels from the left edge of the starting character to the right edge of the last character in the string. Any modification to the inter-character spacing due to a previous call to the AddTextSpace() function is automatically taken into account. The ASCII string is terminated with a NULL character.

Note: If a character code in the string is not defined in the font then the width of the 'undefined character' is used for that character code.

Syntax int32 GetWidth(char* s)

s
ASCII character string.

Return value Returns the width in pixels of the specified ASCII string if drawn using the currently selected text font.

Example This example draws an image that has white cross lines with red text centered between the two left and right sides of the screen.

```
void Test_GetWidth( TGC* gc )
{
  char *s[] =
  {
    "The GetWidth() function",
    "is easily used",
    "on your screen."
  };
  INT32 x, y, i;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the color to be white
  gc->Transparent( TStdColor::white );

  //*** Crosshairs
  gc->DrawLine( 320,   0,   320, 2*240 );
  gc->DrawLine( 0,   240, 2*320,   240 );

  y = 240 - (4 * gc->CharHigh()/2) + gc->GetAscent();
```

```
  //*** Set the color to be red
  gc->Transparent( TStdColor::red );

  gc->SelectFont( "BIG30" );
  for( i = 0; i < 3; ++i )
  {
    x = 320 - gc->GetWidth(s[i])/2;
    gc->DrawString( x, y, s[i] );
    y += gc->CharHigh();
  }
}
```

GetWordSpace

Class Text

Description Gets the horizontal spacing between words. Associated with each text font is a default spacing between a word and the character to its right.

Syntax int32 GetWordSpace(void)

Return value Amount of horizontal space between words. The returned value can be positive or negative.

Example This example draws an image that has a text in which space between the words is increasing.

```
void Test_GetWordSpace( TGC* gc )
{
  INT16 x, y, i;
  char  *s;
  char  s1[32];
  INT32 os = gc->GetWordSpace();

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** set the active color
  gc->Transparent( TStdColor::white );

  gc->SelectFont( "OPIX9" );

  s = "Note increasing spaces between words.";

  for( i = 0, y = 0; i < 20; ++i )
  {
    gc->AddWordSpace(i);
    x = 320 - gc->GetWidth(s)/2;
    y += gc->CharHigh();
    gc->DrawText( x, y, s );
    sprintf( s1, "%d", gc->GetWordSpace() );
    gc->DrawString( x - 40, y, s1 );
  }

  gc->AddWordSpace( os );
}
```

GroundFillVuport

Class Screen fills

Description Sets the entire extent of the active portion of memory to the current foreground color.

Syntax GroundFillVuport(void)

Example This example draws an image with a red background then fills it with black.

```
void Test_GroundFillVuport( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Fill the screen with red
  gc->ColorFillVuport( TStdColor::red );

  //*** Fill the screen with black (the background)
  gc->GroundFillVuport();
}
```

InitMatrix

Class 3-D methods

Description Initializes memory representing a 4x4 matrix to the identity matrix. The matrix is represented in row major form (the rows are stored sequentially in memory).

The resulting identity matrix is:

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

Syntax InitMatrix(int32 matrix[ ])

Return value This function is declared as void, meaning that it returns no value.

Example This example draws an image that has a drawing of a hand watch graphic.

```
typedef long FIX;
      static FIX rotation[3] = {0, 0, 0};
      static FIX translat1[3] = {-320, -240, 0};
      static FIX translat2[3] = {320, 240, 0};
      static long xyz[] = {
         320,40,0, 340,240,0, 320,260,0, 300,240,0
      };
      static short connect[4] = {0, 1, 2, 3};
      FIX   matrix[16];
      FIX   verts[12];
      short xy[2][8];
      unsigned short Which = 0;
      long   angle;      /* angle should be long */
      int  ViewA,  ViewB;

      /* If changing the origin it is better to use a temporary
      viewport */
      ViewA = CurrentVuport();
      ViewB = OpenVuport();
      SelectVuport(ViewB);

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);
```

```
init_canvas();

/* Clear the screen with black. */
BlackFillVuport();

/* Draw a hand watch */
Int32ToFix(3, translat1, translat1);
Int32ToFix(3, translat2, translat2);
for(;;){
   if(kbhit())break;
   for (angle = 0; angle < 360; ++angle) {
      if(kbhit())break;
      InitMatrix(matrix);
      Translate(matrix, translat1);
      rotation[0] = angle << 16L;
      Rotate(matrix, rotation);
      Translate(matrix, translat2);
      Int32ToFix(12, xyz, verts);
      Transform(matrix, 4, verts);
      VertexToPoint(4, verts, xy[Which]);
      Which = Which ^ 1;
      WaitFrames(5);
      Opaque(black,black);
      EncloseFillRect(4, xy[Which]);
      Which = Which ^ 1;
      Transparent(red);
      DrawOval(420, 420, 110, 30);
      Transparent(blue);
      FillConvex(4, connect, xy[Which]);
      Which = Which ^ 1;
   }
}
SelectVuport(ViewA);
CloseVuport(ViewB);
```

InstallPatn

Class Pattern management and primitives

Description Installs a pattern in the pattern table. The 256 bits of the 16x16 two-dimensional bitmap are copied into the pattern table location indicated by *index*. This pattern becomes the current pattern and it is selected for subsequent filling operations. When the installed pattern is later drawn to the screen, the 0s in the pattern are replaced with the current COLOR0, and the 1s in the pattern are replaced with the current COLOR1. The maximum number of patterns n that can be simultaneously installed is obtained from the GetPatnMax() function.

Syntax int32 InstallPatn(int32 index, int16 pattern[ ])

index
Index to be assigned to the pattern.

pattern[]
Pointer to the pattern bitmap.

Return value A value of -1 is returned if index is out of range; otherwise, a value of 0 is returned.

Example This example draws an image that has a rectangle filled with a custom 'X' pattern.

```
void Test_InstallPatn( TGC* gc )
{
  // The original bitmap of the pattern
  /*
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,
  0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,
  0,1,1,1,1,1,0,0,0,0,1,1,1,1,1,0,
  0,0,1,1,1,1,1,0,0,1,1,1,1,1,0,0,
  0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,
  0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,
  0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,
  0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,
  0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,
  0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,
  0,0,1,1,1,1,1,1,0,0,1,1,1,1,1,1,0,0,
  0,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,0,
  0,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,
  0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
  */

  //*** Change the bitmap to an array of INT16's with the left most bit
in the
  //*** bitmap is the MSB in the INT16
```

```
  INT16 mypatn[] =
  {
     0x0000,
     0x1008,
     0x381c,
     0x7c3e,
     0x3e7c,
     0x1ff8,
     0x0ff0,
     0x07e0,
     0x07e0,
     0x0ff0,
     0x1ff8,
     0x3e7c,
     0x7c3e,
     0x381c,
     0x1008,
     0x0000
  };

  //*** Setup look-up-table for colors white and black and colors red,
green
  //*** and blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen with black.
  gc->BlackFillVuport();

  //*** Install (and select) our custom pattern
  //*** Could also select with gc->SelectPatn( "User" );
  gc->InstallPatn ( mypatn );
  gc->Opaque      ( TStdColor::blue, TStdColor::red);
  gc->PatnFillRect( 448, 288, 96, 96);
}
```

Int16ToFix

Class Fixed point methods

Description Converts an array of short integers to fixed- point format. Elements of input array are 16-bit two's complement integers (C type short). Elements of output array are 32-bit two's complement fixed-point numbers whose 16 LSBs are to the right of the binary point. The conversion to fixed-point format is done by shifting the elements left by 16.

Syntax FIX* Int16ToFix(long n, long in_array[ ], FIX out_array[ ])

n
Number of elements to be converted.

in_array
Pointer to input array (elements are integers).

out_array
Pointer to output array (elements are fixed point).

Return value Returns a pointer to the first element of the output array.

Example This example draws an image that has a growing triangle.

```
typedef long FIX;
      static short ptlist[] = { 0,-20, 30,15, -30,15 };
      static short triangle[] = { 0,1, 1,2, 2,0 };
      FIX xy[6];
      int i, j;
      int ViewA, ViewB;

      /* If changing the origin it is better to use a temporary
      viewport */
      ViewA = CurrentVuport();
      ViewB = OpenVuport();
      SelectVuport(ViewB);

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();
```

```
SetOrigin(320, 240);
Transparent(green);
Int16ToFix(6, ptlist, xy);

/* Draw a growing triangle filled with green */
for (j = 0; j < 60; ++j) {
   for (i = 0; i <= 5; ++i)
      xy[i] += xy[i] >> 4L;
   FixToInt16(6, xy, ptlist);
   DrawPolyLine(3, triangle, ptlist);
}
SelectVuport(ViewA);
CloseVuport(ViewB);
```

Int32ToFix

Class Fixed point methods

Description Converts an array of long integers to fixed-point format. Elements of input array are 32-bit two's complement integers (C type long). Elements of output array are 32-bit two's complement fixed-point numbers consisting of 16 bits to the left of binary point, and 16 bits to the right of binary point. The conversion to fixed-point format is done by shifting the elements left by 16.

Syntax FIX *Int32ToFix(long n, long in_array[ ], FIX out_array[ ])

n
Number of elements to be converted.

in_array
Pointer to input array (elements are integers).

out_array
Pointer to output array (elements are fixed point).

Return value Returns a pointer to the first element of the output array.

Example This example draws an image that has a red triangle.

```
typedef long FIX;
static short triangle[] = {0, 1, 2};
static long xyz1[9] = {0,-116,0, 100,58,0, -100,58,0};
FIX     xyz2[9];
short   xy[6];

/* Establish number of image versions. */
SetVersions(0);

/* Setup look-up-table for colors white and black and colors red,
green and blue */
SetLut(BW|HUERGB);

init_canvas();

/* Clear the screen with black. */
BlackFillVuport();

SetOrigin(320, 240);
Int32ToFix(9, xyz1, xyz2);
VertexToPoint(3, xyz2, xy);
Transparent(red);
FillConvex(3, triangle, xy);
```

MoveRect

Class Bitmap icons

Description Copies from one rectangular region of the screen to another.

Syntax MoveRect(int32 width, int32 height, int32 xs, int32 ys, int32 xd, int32 yd)

width
Width of the rectangle.

height
Height of the rectangle.

xs, ys
Coordinates of the top left corner of the source rectangle. If a portion of the source rectangle lies in negative x and y coordinate space, that portion is not copied; only the portion lying in positive xy space is moved.

xd, yd
Coordinates of the top left corner of the destination rectangle. Only the portion of the destination rectangle lying within the current visibility rectangle (the window corresponding to the intersection of the viewport and clipping rectangle) is modified on the screen. The rectangle is copied correctly even in the case in which the source and destination rectangles overlap.

Example This example uses the MoveRect() function to move a rectangle from the upper left corner to the lower right corner of the screen.

```
void Test_MoveRect( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->Opaque( TStdColor::blue, TStdColor::red );

  gc->SelectPatn( "CHECKER8" );
  gc->PatnFillRect( 100, 100, 0, 0 );
  gc->MoveRect( 90, 90, 5, 5, 300, 300 );

  gc->Opaque( TStdColor::white, TStdColor::black );
  gc->DrawRect( 92, 92, 4, 4 );
  gc->DrawRect( 92, 92, 299, 299 );
}
```

MoveVuport

Class Windows

Description Moves the viewport to a new position.

Syntax int32 MoveVuport(int32 x, int32 y)

x, y
Coordinates of the new position for the top left corner of the vuport, given as displacements from the screen origin located at the top left corner of the screen.

Example This example produces an image that has multiple rectangles with text inside.

```
static char *s[] = {
            "Coordinate origin",
            "and clipping rectangle",
            "move with viewport."
      };
      int i, xtext, ytext, xvu, yvu;

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      SetColor0(red);
      for (xvu = yvu = 0; xvu < 447; xvu += 8, yvu += 6) {
            MoveVuport(xvu, yvu);
            SetColor1(blue);
            FillRect(192, 76, 0, 0);
            SetColor1(green);
            DrawRect(190, 74, 1, 1);
            SetColor1(white);
            xtext = CharWideMax();
            ytext = 12 + GetAscent();
            for (i = 0; i <= 2; ++i) {
                  DrawString(xtext, ytext, s[i]);
                  ytext += CharHigh();
            }
      }
```

Opaque

Class Transparency

Description Sets the foreground color and the background color. It also sets the drawing replacement mode. Everything drawn will cover existing colors. Opaque should be used instead of transparent only when text or a pattern is wanted to completely cover what is underneath. This function requires two uint32 color values where COLOR1 represents the foreground and COLOR0 represents the background.

Syntax Opaque (uint32 color1, uint32 color0)

color1
Foreground color.

color0
Background color.

Example This example draws an image that has yellow text with a cyan background.

```
void Test_Opaque( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUECMY );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the Font
  gc->SelectFont( "BIG30" );

  if( TStdColor::yellow && TStdColor::cyan )
  {
    gc->Opaque( TStdColor::yellow, TStdColor::cyan );
    gc->DrawString( 100, 100, "Yellow text with Cyan background" );
  }
}
```

OpenVuport

Class Windows

Description Opens a new viewport and returns an index that is used to identify the viewport in subsequent transactions. The new viewport inherits the attributes of the active viewport at the time of the call. After the attributes have been passed to the new viewport, the new viewport in turn becomes the active viewport. The following attributes are inherited by the new viewport:

- pattern index
- font index
- horizontal text spacing increment
- pen width and height
- viewport width, height and screen position
- viewport x-y origin
- clipping rectangle
- colors 0 and 1
- pixel processing operation
- transparency mode (on or off)
- plane mask

Syntax int32 OpenVuport(void)

Return value If the maximum number of viewports is already open at the time of the call, a value of -1 is returned in place of a valid viewport index.

Example This example draws an image that has a pattern filled rectangle.

```
int vindex;

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();
```

```
SelectPatn(CHECKER8);
/* Viewport 0's color0 = blue */
SetColor0(blue);
/* Open vuport 1 */
vindex = OpenVuport();

/* Viewport 1 inherits color0 from viewport 0 */
/* Viewport 1's color1 = green */
SetColor1(green);

/* Fill square with blue-and-green pattern */
PatnFillRect(96, 96, 272, 192);
```

PatnFillOval

Class Pattern management and primitives

Description Fills an ellipse with a pattern. The ellipse is in standard position, with its major and minor axes parallel to the coordinate axes. The ellipse is defined by the minimum enclosing rectangle in which it is inscribed.

Syntax PatnFillOval (int32 width, int32 height, int32 x, int32 y)

width
Width of the enclosing rectangle.

height
Height of the enclosing rectangle.

xleft, ytop
Coordinates of the top left corner of the enclosing rectangle.

Example This example draws an image that has multiple pattern filled ellipses of different sizes.

```
void Test_PatnFillOval( TGC* gc )
{
  INT32  w, h, x, y, patn;
  UINT32 hue;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();
  gc->TranspOFF();

  //*** Pattern fill ellipses of various sizes
  x = y = hue = 0;
  patn = 1;
  for( w = 640, h = 480; w >= 192; w -= 32, h -= 24 )
  {
    gc->Set_color0( hue += 0x01010101 );
    gc->Set_color1( ~hue );
    gc->SelectPatn( patn++ );
    gc->PatnFillOval( w, h, x, y );
  }
}
```

PatnFillPolygon

Class Pattern management and primitives

Description Fills a polygon with a pattern given a list of points representing the vertices of the polygon. No restrictions are placed on the shape of the polygons filled by the function: edges can cross each other, filled areas can contain holes. The polygon is filled with the current pattern in colors COLOR0 and COLOR1.

Syntax PatnFillPolygon(int32 n, TQD_Point ptlist[ ])

n

Number of vertices in the polygon.

ptlist[]

An array of type IntPoint2, which is a structure of the short x and y coordinates of the vertices. The coordinates of the vertices are at (point[i].x, point[i].y) for i in [0..nvert-1].

Example This example draws an image that has a pattern filled star-shaped polygon and another inside solid-filled polygon.

```
int n = 5;
      IntPoint2 *points, *points1;

      points = (IntPoint2 *)malloc(n*sizeof(IntPoint2));
      points1 = (IntPoint2 *)malloc(n*sizeof(IntPoint2));

      points[0].x = 50;
      points[0].y = 240;
      points[1].x = 620;
      points[1].y = 130;
      points[2].x = 180;
      points[2].y = 450;
      points[3].x = 320;
      points[3].y = 50;
      points[4].x = 420;
      points[4].y = 420;

      points1[0].x = 340;
      points1[0].y = 190;
      points1[1].x = 340;
      points1[1].y = 290;
      points1[2].x = 300;
      points1[2].y = 320;
      points1[3].x = 240;
      points1[3].y = 320;
      points1[4].x = 280;
      points1[4].y = 210;
```

```
/* Establish number of image versions. */
SetVersions(0);

/* Setup look-up-table for colors white and black and colors red,
green and blue */
SetLut(BW|HUERGB);

init_canvas();

/* Clear the screen with black. */
BlackFillVuport();

Opaque(green, white);
SelectPatn(CHECKER8);
PatnFillPolygon(n,points);


FillPolygon(n,points1);


Transparent(red);
DrawLine(points[0].x, points[0].y, points[1].x, points[1].y);
DrawLine(points[1].x, points[1].y, points[2].x, points[2].y);
DrawLine(points[2].x, points[2].y, points[3].x, points[3].y);
DrawLine(points[3].x, points[3].y, points[4].x, points[4].y);
DrawLine(points[4].x, points[4].y, points[0].x, points[0].y);

DrawLine(points1[0].x, points1[0].y, points1[1].x,
points1[1].y);
DrawLine(points1[1].x, points1[1].y, points1[2].x,
points1[2].y);
DrawLine(points1[2].x, points1[2].y, points1[3].x,
points1[3].y);
DrawLine(points1[3].x, points1[3].y, points1[4].x,
points1[4].y);
DrawLine(points1[4].x, points1[4].y, points1[0].x,
points1[0].y);

free(points);
free(points1);
```

PatnFillRect

Class Pattern management and primitives

Description Fills a rectangle with a pattern. The rectangle is pattern-filled with the current pattern, which is drawn in colors COLOR0 and COLOR1. The rectangle is solid-filled with the current COLOR1.

Syntax PatnFillRect(int32 width, int32 height, int32 xleft, int32 ytop)

width
Width of the rectangle.

height
Height of the rectangle.

xleft, ytop
Coordinates of the top left corner of the rectangle.

Example This example draws an image that has 5 pettern filled rectangles.

```
void Test_PatnFillRect( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->Opaque( TStdColor::red, TStdColor::green );
  gc->SelectPatn( "CHECKER8" );
  gc->PatnFillRect( 440, 280, 100, 100 );

  gc->Opaque( TStdColor::white, TStdColor::blue );
  gc->SelectPatn( "MEME" );
  gc->PatnFillRect( 420,  30, 110, 110 );
  gc->PatnFillRect( 220, 220, 110, 150 );
  gc->PatnFillRect( 190, 150, 340, 150 );
  gc->PatnFillRect( 190,  60, 340, 310 );
}
```

PatnFillTriangle

Class Pattern management and primitives

Description Fills a triangle with a pattern. The triangle is pattern-filled with the current pattern, which is drawn in colors COLOR0 and COLOR1.

Syntax PatnFillTriangle(int32 x0, int32 y0, int32 x1, int32 y1, int32 x2, int32 y2)

x0, y0
Coordinates of vertex 0.

x1, y1
Coordinates of vertex 1.

x2, y2
Coordinates of vertex 2.

Example This example draws an image that has 3 pattern filled triangles.

```
void Test_PatnFillTriangle( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->Opaque( TStdColor::white, TStdColor::blue );
  gc->SelectPatn(  "CHECKER8" );
  gc->PatnFillTriangle( 100, 100, 500, 100, 320, 400 );

  gc->Opaque( TStdColor::green, TStdColor::red );
  gc->SelectPatn( "BARS_H2" );
  gc->PatnFillTriangle( 160, 160, 420, 160, 320, 320 );

  gc->Opaque( TStdColor::black, TStdColor::white );
  gc->SelectPatn( "BARS_V2" );
  gc->PatnFillTriangle( 240, 220, 340, 220, 320, 260 );
}
```

PatnFrameOval

Class Pattern management and primitives

Description Fills an ellipse-shaped frame with a pattern. The frame consists of a filled region between two concentric ellipses. The frame is filled with the current pattern in colors COLOR0 and COLOR1. The portion of the screen enclosed is not altered. The outer oval is described by the minimum enclosing rectangle in which it can be circumscribed.

Syntax PatnFrameOval(int32 width, int32 height, int32 xleft, int32 ytop, int32 dx, int32 dy)

width
Width of the enclosing rectangle.

height
Height of the enclosing rectangle.

xleft, ytop
Coordinates of the top left corner of the enclosing rectangle.

dx
Horizontal distance between outer and inner ellipses.

dy
Vertical distance between the outer and inner ellipses.

Example This example demonstrates the PatnFrameOval() function.

```
void Test_PatnFrameOval( TGC* gc )
{
  INT32 i;

  //*** Setup look-up-table
  gc->SetLut( GRAYACE | HUERGB | HUECMY | HUEPEN );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Select a pattern by name
  gc->SelectPatn( "checker1" );

  //*** Draw some ovals
  gc->Set_color0( TStdColor::black );
  for( i = 0; i <= 10; i++ )
  {
    gc->Set_color1( gc->RepPixel( i+1 ) );
    gc->PatnFrameOval( 640 - i*60, 20 + i*46, i*30, 230 - i*23, 8,
    8 );
  }
}
```

PatnFrameRect

Class Pattern management and primitives

Description Fills a rectangular-shaped frame with a pattern. The frame consists of a filled region between two concentric rectangles. The frame is filled with the current pattern in colors COLOR0 and COLOR1. The portion of the screen enclosed is not altered.

Syntax PatnFrameRect(int32 width, int32 height, int32 xleft, int32 ytop, int32 dx, int32 dy)

width
Width of the rectangle.

height
Height of the rectangle.

xleft, ytop
Coordinates of the top left corner of the rectangle.

dx
Horizontal distance between outer and inner rectangle.

dy
Vertical distance between the outer and inner rectangle.

Example This example draws an image that has a red and green pattern-framed rectangle.

```
void Test_PatnFrameRect( TGC* gc )
{
  INT32 w, h, x, y, dx, dy;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the active color
  gc->Opaque( TStdColor::red, TStdColor::green );

  w  = 480;
  h  = 360;
  x  =  80;
  y  =  60;
  dx =  40;
  dy =  30;

  gc->SelectPatn( "CHECKER8" );
  gc->PatnFrameRect( w, h, x, y, dx, dy );
}
```

Perspec

Class 3-D methods

Description Performs perspective transformations on a list of 3D vertices, mapping them to a list of 2D points.

Syntax Perspec(int32 n, int32 vertlist[ ], TQD_Point ptlist[ ], int32 xview, int32 yview, int32 zview)

n

The number of vertices in *vertlist[]*.

vertlist[]

List of vertices. Each 3D vertex in the *vertlist* is represented as a 96-bit quantity consisting of three 32-bit fixed point values: x, y and z. Each fixed point number has 16 bits to the right of the binary point, and 16 bits to the left.

ptlist[]

Output point list. Each 2D point in the *ptlist[]* consists of two short integers: 16 bits for x, and 16 bits for y.

xview, yview, zview

Viewer's reference position used for the perspective transformation.

Example This example draws an image that has a rotating box.

```
typedef long FIX;
      static FIX rotation[3] = { 0, 0, 0 };
      static long xyz[] = {
         -60,-60,-60,  60,-60,-60,
          60, 60,-60, -60, 60,-60,
         -60,-60, 60,  60,-60, 60,
          60, 60, 60, -60, 60, 60
      };
      static short faces[6][4] = {
         { 0, 1, 2, 3 }, { 7, 6, 5, 4 },
         { 4, 5, 1, 0 }, { 5, 6, 2, 1 },
         { 6, 7, 3, 2 }, { 4, 0, 3, 7 }
      };
      FIX   matrix[16], verts[24];
      short xy[2][16];
      unsigned long  i, c, angle;
      unsigned short Which = 0;
      unsigned long frame = wait_frame(0)+1;
      int  ViewA,  ViewB;

      /* If changing the origin it is better to use a temporary
      viewport */
      ViewA = CurrentVuport();
```

```
ViewB = OpenVuport();
SelectVuport(ViewB);

/* Establish number of image versions. */
SetVersions(0);

/* Setup look-up-table for colors white and black and colors red,
green and blue */
SetLut(BW|HUERGB);

init_canvas();

/* Clear the screen with black. */
BlackFillVuport();

/* Center origin */
SetOrigin(320, 240);
/* Draw a rotating box around the y axis */
for(;;){
   if(kbhit())break;
   for (angle = 0; angle < 360; angle += 2) {
         if(kbhit())break;
         InitMatrix(matrix);
         rotation[2] = angle << 16L;
         Rotate(matrix, rotation);
         Int32ToFix(24, xyz, verts);
         Transform(matrix, 8, verts);
         Perspec(8, verts, xy[Which], 0, -80, -200);
         Which = Which ^ 1;
         WaitFrame(frame);
         frame+=1;
         Opaque(black,black);
         EncloseFillRect(8, xy[Which]);
         Which = Which ^ 1;
         for (i = c = 0; i <= 5; ++i) {
            SetColor1(c += 0x11111111);  /* pixel=4 bits */
            FillConvex(4, faces[i], xy[Which]);
         }
         Which = Which ^ 1;
   }
}
SelectVuport(ViewA);
CloseVuport(ViewB);
```

PixelExpand

Class Bitmap icons

Description Moves the contents of the specified packed pixel array into a specified rectangular area on the screen. The destination rectangle is clipped. Only the portion of the rectangle lying within the current visibility rectangle (the intersection of the viewport and clipping rectangle) is altered.

Syntax PixelExpand(int16 srcarray[ ], int32 srcpitch, int32 srcpsize, int32 width, int32 height, int32 xdst, int32 ydst)

srcarray[]
Pointer to the source array.

srcpitch
Pitch of the source array. The pitch is the difference in memory addresses of the start of two adjacent rows of the source array. The pitch must be a positive multiple of the pixel size. The minimum pitch value is (width * srcpsize), where srcpsize is the pixel size.

srcpsize
Bits/pixel of the source array.

width
Width of the destination rectangle.

height
Height of the destination rectangle.

xdst, ydst
Coordinates of the top left corner of the destination rectangle.

Example This example draws an image that has a shape surrounded by copies of itself.

```
int   w, h, pitch, spsize, i;
      long x, y;
      short *buf;
      buf = (short *)malloc(80*60*4/8);

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
```

```
BlackFillVuport();

w = 80;
h = 60;
x = 280;
y = 210;
pitch = w * GetPsize();
spsize = 4;

Transparent(red);
FillRect(w-2, h-2, x+1, y+1);
Opaque(green,blue);
SelectPatn(CHECKER8);
PatnFrameOval(w-2, h-2, x+1, y+1, 20, 15);

GetRect(w, h, x, y, buf, pitch);

for (i = 25, x = 0L, y = -150L<<16L; i > 0; --i) {
      x += y >> 2L;
      y -= x >> 2L;
      PixelExpand(buf, pitch, spsize, w, h,
      (x>>16L)+280, (y>>16L)+210);
}
free(buf);
```

PokeBlackColors

Class Color

Description Sets the intensities of red, green, and blue primaries of the currently selected color code in an analog look-up-table to zero level.

Syntax PokeBlackColors(void)

Example This example draws an image that has rectangles filled with various shades of red and blue.

```
void Test_PokeBlackColors( TGC* gc )
{
  INT32 w, x, i;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** set the active color
  gc->Transparent(TStdColor::red);

  w = gc->width / 5;
  x = (gc->width - (w*5)) / 2;
  for(i=0; i<=4; i++)
  {
    gc->Transparent( gc->RepPixel(i) );
    gc->PokeColor( i );
    gc->PokeLevelRGB( i*250, 0, i*250 ); // mix of red and blue colors
    gc->FillRect( w, gc->height, x, 0 );
    x = x + w;
  }

  //*** Set the intensities of red, green, and blue primaries of the
  //*** currently selected color code in an analog look-up-table to
  //*** zero level.
  gc->PokeBlackColors();
}
```

PokeCode

Class Color

Description Sets DAC code directly at the current color address of the digital look-up-table.

Syntax PokeCode(int32 code)

code
0 to 255.

Example This example demonstrates the PokeCode function.

```
PokeColor(1);
PokeCode(255);
```

PokeColor

Class Color

Description Directly loads a DAC color code into the look-up table (LUT).

Syntax PokeColor(uint32 colorset)

colorset
Color code (0-255).

Example This example draws an image that has rectangles filled with various shades of red and blue.

```
void Test_PokeColor( TGC* gc )
{
  INT32 w, x, i;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the active color
  gc->Transparent( TStdColor::red );

  w = gc->width / 5;
  x = (gc->width - (w*5)) / 2;
  for(i=0; i<=4; i++)
  {
    gc->Transparent( gc->RepPixel( i ) );
    gc->PokeColor( i );
    gc->PokeLevelRGB( i*250, 0, i*250 ); // mix of red and blue colors
    gc->FillRect( w, gc->height, x, 0 );
    x = x + w;
  }
}
```

PokeLevelRGB

Class Color

Description Sets the intensities of red, green, and blue primaries of the currently selected color code in an analog look-up-table (LUT).

0-255 for the rest (GX,GX, SL).

Syntax PokeLevelRGB(unsigned short red, unsigned short green, unsigned short blue)

red
Intensity of red color code.

green
Intensity of green color code.

blue
Intensity of blue color code.

Example This example demonstrates the PokeLevelRGB function.

```
int w, x, i;

      /* Establish number of image versions. */
      SetVersions(0);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

       w=width/5;
       x=(width-(w*5))/2;
       for(i=0;i<=4;i++)
       {
          Transparent(RepPixel(i));
          PokeColor(i);
          PokeLevelRGB(i*250,0,i*250);     /* Testing on GF */
          FillRect(w,height,x,0);
        x=x+w;
       }
```

PutPixel

Class Bitmap icons

Description Writes the specified pixel value to the specified coordinates. Given a pixel size of n bits, the pixel is contained in the n LSBs of the *value* input argument (the 32-n MSBs are ignored).

Syntax PutPixel(int32 value, int32 xd, int32 yd)

value
Source pixel value.

xd, yd
Coordinates of the destination pixel. The coordinates are specified relative to the viewport x-y origin.

Example This example draws an image that has 4 boxes with flipped text inside.

```
void Test_PutPixel( TGC* gc )
{
  char *s[] =
  {
    "Flip all", "the pixels", "in the box."
  };
  INT32 i, j, w, x1, y1, x2, y2;
  TColor val;

  //*** Setup look-up-table for colors white and black and colors red,
green
  //*** and blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen with black.
  gc->BlackFillVuport();

  gc->SelectFont( "OPIX9" );

  //*** Draw picture to be flipped
  gc->Transparent( TStdColor::white );
  y1 = 98;
  for( i = 0; i <= 2; ++i, y1 += gc->CharHigh() )
    gc->DrawString(126, y1, s[i]);
  gc->Opaque( TStdColor::blue, TStdColor::green );
  gc->SelectPatn( "CHECKER8" );
  w = 114;
  x1 = 114;
  y1 = 53;
  gc->PatnFrameRect( w, w, x1, y1, 5, 14 );
```

```
  //*** Now use PutPixel function to flip pixels
  x2 = x1 + 320;
  y2 = y1 +240;
  for( i = 0; i <= 113; ++i )
  {
    for( j = 0; j<= 113; ++j )
    {
      val = gc->GetPixel( x1+i, y1+j );
      gc->PutPixel( val, x2+w-j, y1+i   );
      gc->PutPixel( val, x2+w-i, y2+w-j );
      gc->PutPixel( val, x1+w-j, y2+w-i );
    }
  }
}
```

PutRect

Class Bitmap icons

Description Moves the contents of the specified packed pixel array into a specified rectangular area on the screen. The destination rectangle is clipped. Only the portion of the rectangle lying within the current visibility rectangle (the intersection of the viewport and clipping rectangle) is altered.

Syntax PutRect(int32 srcarray[ ], int32 srcpitch, int32 width, int32 height, int32 xdst, int32 ydst)

srcarray[]
Pointer to the source array.

srcpitch
Pitch of the source array. The pitch is the difference in memory addresses of the start of two adjacent rows of the source array. The pitch must be a positive multiple of the pixel size. The minimum pitch value is (width * psize), where psize is the pixel size.

width
Width of the destination rectangle.

height
Height of the destination rectangle.

xdst, ydst
Coordinates of the top left corner of the destination rectangle.

Example This example draws an image that has a shape surrounded by copies of itself.

```
int   w, h, pitch, i;
      long x, y;
      short *buf;
      buf = (short *)malloc(80*60*4/8);

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      w = 80;
      h = 60;
```

```
x = 280;
y = 210;
pitch = w * GetPsize();

Transparent(red);
FillRect(w-2, h-2, x+1, y+1);
Opaque(green,blue);
SelectPatn(CHECKER8);
PatnFrameOval(w-2, h-2, x+1, y+1, 20, 15);

GetRect(w, h, x, y, buf, pitch);

for (i = 25, x = 0L, y = -150L<<16L; i > 0; --i) {
      x += y >> 2L;
      y -= x >> 2L;
      PutRect(buf, pitch, w, h, (x>>16L)+280, (y>>16L)+210);
}
free(buf);
```

RemovePatn

Class Pattern management and primitives

Description Installs 0s 16 x16 bitmap into the last slot (USER_PATN) in the pattern table.

Syntax Int32 RemovePatn(int32 index)

index
The index of the pattern to be removed.

Return value A value of -1 is returned if index is out of range; otherwise, a value of 0 is returned.

Example This example draws an image that has a solid and a pattern filled rectangle.

```
void Test_RemovePatn( TGC* gc )
{
  // The origina l bitmap of the pattern
  /*
     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,
     0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,
     0,1,1,1,1,1,0,0,0,0,1,1,1,1,1,0,
     0,0,1,1,1,1,1,1,0,0,1,1,1,1,1,0,0,
     0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,
     0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,
     0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,
     0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,
     0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,
     0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,
     0,0,1,1,1,1,1,1,0,0,1,1,1,1,1,1,0,0,
     0,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,0,
     0,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,
     0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,
     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
   */

  //*** Change the bitmap to array of shorts with the left most bit in
the
  //*** bitmap is the MSB in the short
  INT16 mypatn[] =
  {
     0x0000,
     0x1008,
     0x381c,
     0x7c3e,
     0x3e7c,
     0x1ff8,
     0x0ff0,
```

```
    0x07e0,
    0x07e0,
    0x0ff0,
    0x1ff8,
    0x3e7c,
    0x7c3e,
    0x381c,
    0x1008,
    0x0000
  };

  //*** Setup look-up-table for colors white and black and colors red,
green
  //*** and blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen with black.
  gc->BlackFillVuport();

  //*** Assign user index = USER_PATN
  gc->InstallPatn ( mypatn );
  gc->Opaque      ( TStdColor::blue, TStdColor::red );
  gc->PatnFillRect( 100, 100, 10, 10 );

  //*** Fill in the bitmap with 0s, so color0 is the
  //*** one which will be displayed
  gc->RemovePatn();
  gc->PatnFillRect( 100, 100, 150, 150 );
}
```

RemoveVuport

Class Windows

Description Removes (closes) the viewport identified by the specified index. The viewport must be open at call. Viewport 0 can never be removed.

Syntax int32 RemoveVuport(int32 index)

index
Index of the viewport to be removed.

Return value When the function is called with a valid index, the value 0 is returned to confirm that the viewport was removed as requested. When the function is called with an invalid index, a value of -1 is returned to indicate that no action was taken.

Example This example draws an image that has two pattern-filled rectangles.

```
int vindex;

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      SelectPatn(CHECKER8);
      /* Viewport 0's color0 = red and color1 = blue */
      Opaque(blue,red);
      /* Open vuport 1 */
      vindex = OpenVuport();

      /* Viewport 1 inherits color0 (red) from viewport 0 */
      /* Viewport 1's color1 = green not blue */
      SetColor1(green);

      /* Fill square with red-and-green pattern */
      PatnFillRect(96, 96, 272, 192);

      /* Green is not color1 anymore, it is blue */
      RemoveVuport(vindex);
      PatnFillRect(96, 96, 272, 192);
```

RepPixel

Class Color

Description Replicates a pixel throughout a 32-bit integer. The output of this function is in suitable form to be used as input to the SetColor0(), SetColor1(), or Set_Pmask() functions.

For example, given a pixel size of 4 and an input value of 5, the return value is 0x55555555. An input value of 0x01234567 produces a return value of 0x77777777, and so on.

Syntax uint32 RepPixel(uint32 pixelvalue)

pixelvalue

Return value Given a pixel size of n bits, then LSBs of the input argument (the 32-n MSBs are ignored) are replicated 32/n times to fill the 32-bit return argument.

Example This example draws an image that has a filled rectangle at the uppet left corner of the screen.

```
/* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      SetColor1(RepPixel(4));
      FillRect(100, 100, 0, 0);
```

ResetMatrix

Class 3-D methods

Description Clears the transformation matrix and resets it to the default values of the origin in upper left hand corner. The normalwidth and normalheight variables are assigned to their normalized values with 1.0 for the minor axis.

Syntax ResetMatrix(void)

Example This example changes the origin of the screen and produces no image.

```
ResetMatrix();
      /* Reset the matrix */
      /* Translate origin to the center of screen */
      TranslateX(normalwidth/2);
      TranslateY(normalheight/2);
      /* Reset the matrix and translate again. When matrix is reset
      and then retranslated the point again appears at center of screen
      instead of lower right corner if it were not reset */
      ResetMatrix();
      TranslateX(normalwidth/2);
      TranslateY(normalheight/2);
```

Rotate

Class 3-D methods

Description Multiplies the specified 4x4 transformation matrix by a rotation matrix created from the three angles given as input arguments. The rotation matrix applies the rotations in the order of the angles; that is, the rotation in the xy plane is performed first, the rotation in the yz plane is performed second, and the rotation in the zx plane is performed third. The xy, yz and zx rotation angles are given in degrees, and are contained in angle[0], angle[1] and angle[2], respectively. The elements of the *matrix[]* and *angle[]* arrays are 32-bit numbers in fixed point format. A positive rotation of 90 degrees in the xy plane moves a point on the positive x axis onto the positive y axis. A positive rotation of 90 degrees in the yz plane moves a point on the positive y axis onto the positive z axis. A positive rotation of 90 degrees in the zx plane moves a point on the positive z axis onto the positive x axis. The following matrices represent the effect on an identity matrix when the rotational values are applied individually for the three axes.

Rotation in XY plane (about the Z axis): Az = angle[0]

cos Az sin Az 0 0

-sin Az cos Az 0 0

0 0 1 0

0 0 0 1

Rotation in YZ plane (about the X axis): Ax = angle[1]

1 0 0 0

0 cos Ax sin Ax 0

0 -sin Ax cos Ax 0

0 0 0 1

Rotation in ZX plane (about the Y axis): Ay = angle[2]

cos Ay 0 -sin Ay 0

0 1 0 0

sin Ay 0 cos Ay 0

0 0 0 1

Syntax Rotate(int32 matrix[], int32 angle[])

matrix[]

angle[]

Example This example draws an image that has a rectangle rotated in each of the 3 dimensions.

```
typedef long FIX;
      static FIX rot[3] = { 0, 0, 0 };
      static long xyz[] = {
         -50,-50,0, 50,-50,0, 50,50,0, -50,50,0
      };
      static short front[4] = { 0,1,2,3 };
      static short back[4] = { 3,2,1,0 };
      FIX matrix[16], verts[12];
      short xy[2][8];
      long   i, angle;    /* angle should be long */
      unsigned short Which = 0;
      unsigned long frame = wait_frame(0)+1;
      int  ViewA,  ViewB;

      /* If changing the origin it is better to use a temporary
      viewport */
      ViewA = CurrentVuport();
      ViewB = OpenVuport();
      SelectVuport(ViewB);

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      /* Center origin */
      SetOrigin(320, 240);
      /* Rotate the square in each of the 3 planes */
      for (i = 0; i <= 2; ++i)
         for (angle = 0; angle <= 360; ++angle) {
            InitMatrix(matrix);
            rot[i] = angle << 16L;
            Rotate(matrix, rot);
            Int32ToFix(12, xyz, verts);
            Transform(matrix, 4, verts);
            Perspec(4, verts, xy[Which], 0, -100, -300);
            Which = Which ^ 1;
            WaitFrame(frame);
            frame+=1;
            Opaque(black,black);
            EncloseFillRect(4, xy[Which]);
            Which = Which ^ 1;
            Transparent(red);
            FillConvex(4, front, xy[Which]);
```

```
        Transparent(blue);
        FillConvex(4, back, xy[Which]);
        Which = Which ^ 1;
    }
SelectVuport(ViewA);
CloseVuport(ViewB);
```

Scale

Class 3-D methods

Description Multiplies the specified 4x4 transformation matrix by a scaling matrix created from the three scaling factors given as input arguments. The scaling factors in the x, y and z dimensions are contained in factor[0], factor[1] and factor[2], respectively. The elements of the *matrix[]* and *factor[]* arrays are 32 bits in fixed point fomart.

Syntax Scale(int32 matrix[ ], int32 factor[ ])

matrix[]

factor[]

Example This example draws an image that has a scaled rectangle in each of the 3 dimensions.

```
typedef long FIX;
      static FIX factor[3] = { 0x8000, 0x8000, 0x8000 };
      static long xyz[] = {
         -50,-50,50, 50,-50,50, 50,50,-50, -50,50,-50
      };
      static short object[4] = { 0,1,2,3 };
      FIX matrix[16], verts[12];
      short xy[2][8];
      long   i, f;   /* f should be long */
      unsigned short Which = 0;
      unsigned long frame = WaitFrame(0)+1;
      int  ViewA,  ViewB;

      /* If changing the origin it is better to use a temporary
      viewport */
      ViewA = CurrentVuport();
      ViewB = OpenVuport();
      SelectVuport(ViewB);

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();
```

```
SetOrigin(320, 240);
/* Scale object in each of the 3 dimensions */
for (i = 0; i <= 2; ++i)
   for (f = 0x8000; f <= 0x18000; f += 0x200) {
      InitMatrix(matrix);
      factor[i] = f;
      Scale(matrix, factor);
      Int32ToFix(12, xyz, verts);
      Transform(matrix, 4, verts);
      Perspec(4, verts, xy[Which], 0, 0, -200);
      Which = Which ^ 1;
      WaitFrame(frame);
      frame+=1;
      Opaque(black,black);
      EncloseFillRect(4, xy[Which]);
      Transparent(red);
      Which = Which ^ 1;
      FillConvex(4, object, xy[Which]);
      Which = Which ^ 1;
   }
SelectVuport(ViewA);
CloseVuport(ViewB);
```

SelectColor

Class Color

Description Sets the current color code for the look-up table (LUT).

Syntax SelectColor(uint32 colorcode)

colorcode
Color code for the LUT (0 to 255).

Example This example draws an image that has vertical bars in different shades of red.

```
void Test_SelectColor( TGC* gc )
{
  INT32 w, x, i;
  INT32 r, g, b;
  INT32 rr, gg, bb;

  //*** Setup look-up-table for colors white and black and colors red,
green
  //*** and blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen with black.
  gc->BlackFillVuport();

  w = gc->width / 256;
  x = (gc->width - (w*256))/2;
  g = b = 0;
  for( i = 0; i <= 255; i++ )
  {
    r = i;
    gc->Transparent( gc->RepPixel(i) );
    gc->SelectColor( i );
    rr = gc->MapLevel(r);
    gg = gc->MapLevel(g);
    bb = gc->MapLevel(b);
    gc->SetLevelRGB( rr, gg, bb );
    gc->FillRect( w, gc->height, x, 0 );
    x = x + w;
  }
}
```

SelectFont

Class Font management

Description Selects the font identified by an index or by name. The designated font is used in all subsequent text-drawing operations within the current viewport until a new font is selected.

Syntax int32 SelectFont(uint32 *index*)

index
Index of font to be selected.

int32 SelectFont(const char* *fontname*)

fontname
Name of font to select.

Return value If the font is not currently open, or the index is invalid, a value of -1 is returned to indicate an error, and the system font is selected as a default. Otherwise, a value of 0 is returned as confirmation.

Example This example draws an image with “hello world” printed in green using font at index 1.

```
void Test_SelectFont( TGC* gc )
{
  INT32  x, y;
  char   s[] = "Hello world";

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->Transparent( TStdColor::green );

  //*** Select font by index
  gc->SelectFont( 1 );

  x = 320 - gc->GetWidth(s)/2;
  y = gc->CharHigh();
  gc->DrawString( x, y, s );
}
```

Comments The table below lists font names and descriptions

Font Name	Font Index	Description
SYS16	0	IBM-type alphanumeric font that has printable characters for ASCII codes 0-128. It uses an 8x16 monospaced character block)
OPIX9	1	Alphanumeric font that has printable characters for ASCII codes 32-126. It uses a 5x7 monospaced character block)
FOCUS12	2	Two characters used for the Focus_Cx and Focus_H test images; ASCII codes = 67 and 72; 7x9 character block)
FOCUSMAC	3	Single character used in the Focus_Oo test image; ASCII code = 79; 8x6 character block)
MEMSONY	4	Single character used for various MEME images; ASCII code = 77; 16x16 MEME character in 18x18 block)
KANJIKAN	5	Single Japanese KanjiKan character used in the KanjiKan test image: ASCII code = 75; 22x22 KAN character in 24x24 block)
FOCUSAT5	6	A single @ character used in the LinFocus image; ASCII code = 64; 6x9 @ character in 8x16 block)
FOCUSAT6	7	A single @ character used in the Focus@6 image; ASCII code = 64; 8x7 @ character in 16x16 block)
FOCUSAT7	8	A single @ character used in the Focus@7 image; ASCII code = 64; 8x8 @ character in 16x16 block)
FOCUSAT8	9	A single @ character used in the Focus@8 image; ASCII code = 64; 11x11 @ character in 16x16 block)
MEMEPLUS	10	A meme plus character used in focus and convergence; ASCII code = 77; 43x49 character in a 49x51 block)
MEMESAMS	11	19x16 MEME character in 21x18 block - character 77=M)
MEMEAPPS	12	19x16 MEME character in 21x18 block - character 77=M)
FOCUSLB5	13	5x5 # character in 7x7 block - character 35=#)
FOCUSLB7	14	6x7 # character in 8x9 block - character 35=#)
FOCUSLB8	15	6x7 # character in 8x9 block - character 35=#)
FOCUSLB9	16	9x10 # character in 12x14 block - character 35=#)
FOCUSHIO	17	11x11 O character in 13x16 block - character 79=O)
ELITE11	18	8x9 font in 9x12 block - characters 0-167)
CHINA24	19	24x24 characters in 26x26 block - characters 65-75)
FOCUSHP7	20	6x7 Cx character in 8x9 block - character 67=C)
THIN9	21	5x7 font in 7x11 block - characters 32-127)

Font Name	Font Index	Description
THIN12	22	7x9 font in 9x12 block - characters 0-127)
MEME1111	23	11x11 MEME character in 12x12 block - character 77=M)
CHINA14	24	15x14 character in 17x16 block - character 97=a)
CDCXSONY	25	16x16 CdCx character in 18x18 block - character 67=C)
PLUSSONY	26	43x31 character in 50x32 block - character 69=E)
FOCUSAT9	27	7x9 @ character in 8x16 block - character 64=@)
BIG30	28	Px30 font in 30x32 block - characters 0-255)
ICON32	29	32x32 characters in 32x32 blocks - - characters 65-75)
ADI16	30	8x16 PC font characters 0-254 with IBM drawing figures 176-223)

SelectFontNamed

Class Font management

Description Selects the font identified by name. The designated font is used in all subsequent text-drawing operations within the current viewport until a new font is selected.

Syntax Int32 SelectFontNamed(char* name)

name
Name of font to be selected.

Return value Returns the index of the named font in the font memory pool. Index returned is suitable for use with the SelectFont() function. If the font is not found, returns -1.

Example This example draws an image that has Hello world printed in blue using font BIG30.

```
void Test_SelectFontNamed( TGC* gc )
{
  INT32  x, y;
  char   s[] = "Hello world";

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->SelectFontNamed( "BIG30" );

  gc->Transparent( TStdColor::blue );
  x = 320 - gc->GetWidth( s )/2;
  y = gc->CharHigh();
  gc->DrawString( x, y, s );
}
```

Comments For a list of font names, see SelectFont(),

SelectPatn

Class Pattern management and primitives

Description Selects the pattern identified by the specified index or name. The pattern is used in all subsequent pattern-filling functions in the current viewpoint until another pattern is selected. The pattern is a 16 x16 bitmap that is bit-expanded to the current COLOR0 and COLOR1 values when drawn to the screen. Runtime initialization loads the pattern table with a number of default patterns.

Syntax int32 SelectPatn(int32 *index*)

index
Index position of pattern in pattern table.

nt32 SelectPatn(const char* *name*)

name
Name of pattern.

Return value If index argument is negative or is greater than or equal to the value returned by the GetPatnMax() function, a value of -1 is returned to flag the error condition. Otherwise, a value of 0 is returned to confirm that the designated pattern was selected.

Example This example draws an image that has a blue and green checker board.

```
void Test_SelectPatn( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->Opaque( TStdColor::green, TStdColor::blue );

  gc->SelectPatn( "CHECKER8" );
  gc->PatnFillRect( 160, 160, 100, 100 );
}
```

Comments The table below lists pattern names and descriptions.

Pattern name	Pattern Index	Description
GRAY0	0	Outline (Transparent)
GRAY7	1	Gray*X* where *X* is an integer between 1 and 99, and indicates the percentage of the pixels that are active.
GRAY13	2	
GRAY19	3	

Pattern name	Pattern Index	Description
GRAY25	4	
GRAY31	5	
GRAY38	6	
GRAY44	7	
GRAY50	8	
GRAY56	9	
GRAY63	10	
GRAY69	11	
GRAY75	12	
GRAY81	13	
GRAY88	14	
GRAY94	15	
GRAY100	16	Solid (Opaque)
CHECKER1	17	Checkerboard alternating one pixel on and one pixel off
CHECKER2	18	Checkerboard alternating two pixels on and two pixels off
CHECKER4	19	Checkerboard alternating four pixels on and four pixels off
CHECKER8	20	Checkerboard alternating eight pixels on and eight pixels off
BARS_V1	21	Vertical bars one pixel wide
BARS_V2	22	Vertical bars two pixels wide
BARS_V4	23	Vertical bars four pixels wide
BARS_V8	24	Vertical bars eight pixels wide
BARS_H1	25	Horizontal bars one pixel wide
BARS_H2	26	Horizontal bars two pixels wide
BARS_H4	27	Horizontal bars four pixels wide
BARS_H8	28	Horizontal bars eight pixels wide
MEME	29	Repeating MEME pattern
FCC_EMI	30	Repeating three pixels on and one pixel off pattern

SelectVuport

Class Windows

Description Selects the viewport identified by the specified index. The viewport must previously have been opened (except for viewport0, which is always open). All subsequent graphics operations refer to the selected viewport until another viewport is selected.

Syntax int32 SelectVuport(int32 index)

index
Index of viewport to be selected.

Return value A value of 0 is returned if the index is valid. In the event of an invalid index, a value of -1 is returned and the active viewport is not changed.

Example This example draws an image that has a pattern-filled oval and a pattern-filled rectangle.

```
int  v;
      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      SetClipRect(320, 480, 0, 0);

      /* Open viewport 1 */
      v = OpenVuport();
      MoveVuport(320, 0);
      Opaque(blue,green);
      SelectPatn(CHECKER8);

      /* Display viewport 0's colors and pattern */
      SelectVuport(0);
      Opaque(red,white);
      SelectPatn(CHECKER8);

      PatnFillOval(300, 460, 10, 10);

      /* Display viewport 1's colors and pattern */
      SelectVuport(v);
      PatnFillRect(300, 460, 10, 10);
```

Set_Pmask

Class Register access

Description Loads a generator's plane mask (PMASK) register with specified value. It specifies the plane mask that is used in subsequent drawing operations. The mask determines which bits in a pixel can be modified during drawing operations. The 0s in the mask enable modification of the corresponding bit planes. The 1s in the mask write-protect the corresponding planes.

The mask size is in principle the same as the pixel size, but it must be replicated through the entire 32-bit mask argument. Given a pixel size of n bits, the n-bit mask value must be replicated 32/n times. For example, at four bits per pixel, a mask value of 6 is replicated 32/4 = 8 times to form the pixel_val argument value 0x66666666.

Syntax Set_Pmask(short mask)

mask

Example This example demonstrates the Set_Pmask function.

```
int  ViewA,  ViewB;
      int i, x, y, dx, objwidth, objheight;
      unsigned long frame = WaitFrame(0)+1;

      /* If changing the origin it is better to use a temporary
      viewport */
      ViewA = CurrentVuport();
      ViewB = OpenVuport();
      SelectVuport(ViewB);

      /* Establish number of image versions. */
      SetVersions(0);

      init_canvas();

      SetLut(BLACK); /* setup LUT for black */

      gray25 = RepPixel(0x01 | 0x04);
      SelectColor(0x1);
      SetLevelGrayPercent(25);
      SelectColor(0x4);
      SetLevelGrayPercent(25);
      SelectColor(0x3);
      SetLevelGrayPercent(25);

      white = RepPixel(0x02 | 0x08);
      SelectColor(0x2);
```

```
SetLevelGrayPercent(100);
SelectColor(0x8);
SetLevelGrayPercent(100);

/* Clear the screen with black. */
BlackFillVuport();

Transparent(gray25);
for(i = 0; i <= 460; i += 40){
   FillRect(20, height-(height/4),
   centerleft-(SquareDX(height)/2)+i, top+(height/8));
}

SetClipRect(SquareDX(height)-20,height-(height/4),
centerleft-(SquareDX(height)/2),top+(height/8));

objwidth = 80;
objheight = 60;

x = centerleft-(SquareDX(height)/2);
y = height/2;
dx = 1;

for(;;){
   if(kbhit())break;
   if (Cpw(x, y) & 1) {     /* Bounce off left wall */
       dx = -dx;
   }
   if (Cpw(x+objwidth, y) & 2) { /* Bounce off right wall */
       dx = -dx;
   }

   Opaque(white,black);
   Set_Pmask(gray25);
   FillRect(objwidth, objheight, x, y);
   WaitFrame(frame);
   frame+=1;
   Opaque(black,black);
   Set_Pmask(gray25);
   FillRect(objwidth, objheight, x, y);
   x += dx;
}
SetClipRect(width, height, left, top);
Set_Pmask(0);
SelectVuport(ViewA);
CloseVuport(ViewB);
```

Set_ppop

Class Register access

Description Defines the pixel processing operation for subsequent drawing operations. The specified Boolean or arithmetic operation determines the manner in which source and destination pixel values are combined.

Syntax Set_ppop(int16 code)

code
Pixel processing code 0 to 21.

Example This example draws an image that draws an anitmated oval for the different ppop operations.

```
static short  x[] = { 300, 500, 20, 123 };
      static short  y[] = { 400, 100, 50, 321 };
      static short dx[] = { 1, 2, 3, 4 };
      static short dy[] = {1, 2, 1, 2 };
      int   ppop, j, i;

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      /* Show effects of ppops 0 through 21 */
      for (ppop = 0; ; ) {
         if(kbhit())break;
         Set_ppop(ppop);
         if (++ppop > 21)
            ppop = 0;
         for (j = 0; j < 1000; ++j){
            if(kbhit())break;
            for (i = 0; i <= 3; ++i) {
               if (Cpw(x[i], y[i]) & 0x3)
                  dx[i] = -dx[i];
               if (Cpw(x[i], y[i]) & 0xc)
                  dy[i] = -dy[i];
               x[i] += dx[i];
               y[i] += dy[i];
               SetColor1(RepPixel(i+1));
               FillOval(80, 60, x[i]-40, y[i]-30);
            }
         }
      }
```

SetClipRect

Class Windows

Description Sets the size and position of the clipping rectangle for subsequent drawing operations. Drawing operations can alter only pixels within the visibility rectangle formed by the intersection of the viewport, the clipping rectangle, and the screen. Only pixels within the visibility rectangle consisting of the intersection of the viewport with the clipping rectangle can be altered by graphics output functions.

Syntax int32 SetClipRect(int32 width, int32 height, int32 left, int32 top)

width
Width of the clipping rectangle.

height
Height of the clipping rectangle.

xleft, ytop
Coordinates of the top left corner of the clipping rectangle, specified as displacements from (xorigin, yorigin), the origin for the current viewport.

Example This example draws an image that has 3 figures.

```
int  ViewA,  ViewB;

      /* If changing the origin it is better to use a temporary
      viewport */
      ViewA = CurrentVuport();
      ViewB = OpenVuport();
      SelectVuport(ViewB);

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      Opaque(green,white);
      SetClipRect(128, 96, 64, 48);
      SelectPatn(CHECKER8);

      /* The area in cliprect is only affected */
      PatnFillRect(1000, 1000, -100, -100);
```

```
/* The upper left corner of cliporect now
coincides and moved with the new origin */
SetOrigin(175, 150);
Opaque(red,white);
SelectPatn(CHECKER8);

/* The part of the oval in the new cliprect is affected */
PatnFillOval(256, 192, -64, -48);

/* The new vuport is set at 160,120 from the new origin
with the cliprect moving to that point */
MoveVuport(160, 120);
Transparent(blue);

/* The area in cliprect is only affected */
FillRect(1000, 1000, -100, -100);
SelectVuport(ViewA);
CloseVuport(ViewB);
```

SetCode

Class Color

Description Set specified code at the current color address of the digital look-up- table. (For GX and GC only; others have no digital look-up-table.)

Syntax SetCodet(unsigned code)

Code
0 to 255.

Example This example demonstrates the SetCode function.

```
SelectColor(1);
SetCode(255);
```

SetColor0

Class Register access

Description Sets look-up table (LUT) color 0 to a specific pixel value. Given a pixel size of n bits, the n-bit pixel value must be replicated throughout the 32-bit input argument 32/n times. For example, given a pixel size of 4 bits and a pixel value of 5, the input argument to the SetColor0() function should be 0x55555555.

Syntax SetColor0(int32 pixel_value)

pixel_value
Pixel value replicated to 32 bits.

Example This example draws an image that has 16 different pattern-filled ovals and a numeric count string drawn in red over a varying background color0.

```
static char *s[] = {
            "0", "1", "2", "3", "4", "5",
            "6", "7", "8", "9", "10",
            "11", "12", "13", "14", "15"
            };
      int i;

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      SelectFont(OPIX9);
      for (i = 0; i <= 15; ++i){
         SelectPatn(i);
         SetColor0(RepPixel(i));
         DrawString(i*40+5, 25, s[i]);
         PatnFillOval(30, 430, i*40+5, 50);
      }
```

SetColor1

Class Register access

Description Sets look-up table (LUT) color 1 to a specific pixel value. Given a pixel size of n bits, the n-bit pixel value must be replicated throughout the 32-bit input argument 32/n times. For example, given a pixel size of 4 bits and a pixel value of 5, the input argument to the SetColor0() function should be 0x55555555.

Syntax SetColor1(int32 pixel_value)

pixel_value
Pixel value replicated to 32 bits.

Example See page 160.

SetColorNamed

Class Color

Description Sets the current color name used for drawing. The color name must be provided using a char array.

Syntax uint32 SetColorNamed(char* name)

name
Color name.

Return value Returns the associated uint32 color value used to represent the named color.

Example This example draws an image that has a red rectangle.

```
void Test_SetColorNamed( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set up the color red at slot 2 and pick it up
  gc->Transparent( gc->SetColorNamed( "red" ) );

  gc->DrawRect( 100, 100, 100, 100 );
}
```

SetEdgeColorNamed

Class Color

Description Sets and creates edge colors for rendering anti-aliased TV images.

Syntax uint32 SetEdgeColorNamed(char* leftname, char* rightname)

leftname
Left color name.

rightname
Right color name.

Return value Returns an uint32 that is used to represent the new edge color.

Example 1
```
Uint32 yellowblack;
yellowblack = SetEdgeColorNamed("yellow", "black");
```

Example 2
```
/* Setup LUT and display anti-aliased blue & yellow bars
side-by-side.*/
int blackblue;
int blueyellow;
int yellowblack;

/* Set up the color black at slot 0 */
SetLut(BLACK);
blue = SetColorNamed("blue");
yellow = SetColorNamed("yellow");
/* Fill next N slots with// black->blue sine-squared transition
samples */
blackblue = SetEdgeColorNamed("black", "blue");
/* Fill next N slots with blue->yellow sine-squared transition
samples */
blueyellow = SetEdgeColorNamed("blue", "yellow");
/* Fill next N slots with yellow->black sine-squared transition
samples */
yellowblack = SetEdgeColorNamed("yellow", "black");
```

Example 3
```
int w, i, j, e;
       unsigned long color[7][2] ;
       char s[32];
       char s1[] = "no of samples ";

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for color black */
```

```
  SetLut(BLACK);

  init_canvas();

  /* Clear the screen with black. */
  BlackFillVuport();

color[0][0] = SetColorNamed("white");
color[0][1] = SetEdgeColorNamed("white","yellow");
color[1][0] = SetColorNamed("yellow");
color[1][1] = SetEdgeColorNamed("yellow","cyan");
color[2][0] = SetColorNamed("cyan");
color[2][1] = SetEdgeColorNamed("cyan","green");
color[3][0] = SetColorNamed("green");
color[3][1] = SetEdgeColorNamed("green","magenta");
color[4][0] = SetColorNamed("magenta");
color[4][1] = SetEdgeColorNamed("magenta","red");
color[5][0] = SetColorNamed("red");
color[5][1] = SetEdgeColorNamed("red","blue");
color[6][0] = SetColorNamed("blue");
color[6][1] = SetEdgeColorNamed("blue","black");
/* Eighth bar is black. */

w=width/8;
e=GetEdgeColorMany();

if( w < (e + 1) ) return(TRUE);

for(i = 0;i < 7;i++)
{
  Transparent(color[i][0]);
     FillRect(w-e,1,(i*w),0);

     for(j = 0;j < e;j++)
     {
        Transparent(GetEdgeColor(color[i][1],j));
        DrawPoint(((i*w)+w-e+j),0);
     }
}

Opaque(black,black);
FillRect(w,1,(7*w),0);

/* Turn transparency off so black areas get filled too */
TranspOFF();
CurtainFillRect(width,height,0,0,0);

SelectFont(BIG30);
itoaf(e,s,10,0,"");
strcat(s1,s);
Transparent(color[0][0]);
DrawItalic(210,240,s1);
```

SetLevelGray

Class Color

Description Sets all primaries of the current color to a specified level of the analog look-up-table (LUT). Note that this function preserves the current PRIMARY setting.

Syntax void SetLevelGray(uint32 level)

level
Intensity level (0 to 255).

Example This example draws an image that has 5 gray rectangles.

```
void Test_SetLevelGray( TGC* gc )
{
  INT32 w, x, i;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | GRAYDEC | GRAYEND | GRAYCON );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** set the active color
  gc->Transparent( TStdColor::red );

  w = gc->width/5;
  x = (gc->width - (w*5))/2;
  for( i = 0; i <= 4; i++ )
  {
    gc->Transparent( gc->RepPixel( i ) );
    gc->SelectColor( i );
    gc->SetLevelGray( i*20 );
    gc->FillRect( w, gc->height, x, 0 );
    x = x + w;
  }
}
```

SetLevelGrayPercent

Class Color

Description Sets the intensity level of the primaries of the current color in the analog look-up-table (LUT). Note that this function preserves the current PRIMARY setting.

Syntax SetLevelGrayPercent(uint32 percent)

percent
Intensity level (0 to 100%).

Example This example draws an image that has 5 gray rectangles.

```
void Test_SetLevelGrayPercent( TGC* gc )
{
  INT32 w, x, i;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | GRAYDEC | GRAYEND | GRAYCON );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** set the active color
  gc->Transparent( TStdColor::red );

  w = gc->width / 5;
  x = (gc->width - (w*5)) / 2;
  for( i = 0; i <= 4; i++ )
  {
    gc->Transparent( gc->RepPixel( i ) );
    gc->SelectColor( i );
    gc->SetLevelGrayPercent( i * 25 );
    gc->FillRect( w, gc->height, x, 0 );
    x = x + w;
  }
}
```

SetLevelMax

Class Color

Description Sets the maximum luminance level. Level 0 is used to represent the minimum luminance level (black), while levelmax represents a maximum luminance level (red, green, yellow, etc.). Level max defaults to 255.

Syntax SetLevelMax(uint32 max)

max
Maximum luminance level (0 to 255).

Example This example demonstrates the SetLevelMax function.

```
int i, x, w;
      char s[32];

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      w=width/16;
      x=(width-(w*16))/2;
      /* level 0 -> level 15 */
      SetLevelMax(15);

      for(i=0;i<=GetLevelMax();i++)
      {
          Transparent(RepPixel(i));
          SelectColor(i);
          SetLevelRGB(0,i,0);
          FillRect(w,height,x,0);
          x=x+w;
       }

      itoaf(GetLevelMax(),s,10,0,"");
      SelectFont(OPIX9);
      /* Use max luminance (green) to write the text */
      Transparent(RepPixel(15));
      DrawString(320,240,s);
```

SetLevelRGB

Class Color

Description Sets the intensities of red, green, and blue primaries of the currently selected color code in an analog look-up-table (LUT). Note that this function preserves the current PRIMARY setting.

Syntax SetLevelRGB(unsigned short r, unsigned short g, unsigned short b)

r, g, b
0 to 255

Example This example demonstrates the SetLevelRGB() function.

```
void Test_SetLevelRGB( TGC* gc )
{
  INT32 w, x, i;
  INT32 r, g, b;
  INT32 rr, gg, bb;

  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** set the active color
  gc->Transparent( TStdColor::blue );

  w = gc->width / 256;
  x = (gc->width - (w*256)) / 2;
  r = g = 0;
  for( i = 0; i <= 255; i++ )
  {
    b = i;
    gc->Transparent( gc->RepPixel( i ) );
    gc->SelectColor( i );
    rr = gc->MapLevel(r);
    gg = gc->MapLevel(g);
    bb = gc->MapLevel(b);
    gc->SetLevelRGB( rr, gg, bb );
    gc->FillRect( w, gc->height, x, 0 );
    x = x + w;
  }
}
```

SetLut

Class Color

Description Initializes both digital and analog look-up tables (LUTs) given a colorset token.

Syntax SetLut(uint32 *colorset*)

colorset

Ones in this bit-mask indicate the color subsets that are desired. The color variables are updated by this function. Color variables are assigned a non-zero value as they are made available. Black is always made available and is always assigned zero.

In most situations all colors cannot be used at the same time, and some or all requested colors may not be given. The color can be verified if the value of color variable is non-zero. If a color is attempted to be drawn when not available, it will draw black.

Example 1
```
// just give me black and white
SetLut(BW);
```

Example 2
```
// controls lut
SetLut(BW | GRAYEND | GRAYDEC | HUERGB | HUECMY);
```

Example 3
```
// SMPTE133
SetLut(BW | GRAYEND | GRAYCON | GRAYDEC | HUERGB | HUECMY);
```

Example 4
```
// graybar lut(16-level)
SetLut(BW | GRAYHEX);
```

Example 5
```
// All 39 available colors
SetLut(BW | HUERGB | HUECMY | HUEMID | GRAYHEX | GRAYCON |
GRAYEND | GRAYDEC);
```

Example 6
```
// All grays and 256 color linear gray scale
SetLut(GRAYALL)
```

Comments If you decide to use a standard setup, you will be limited to using either the standard colors listed in this section or a standard linear grayscale.

The SetLut() function can be used to rapidly setup the LUT and update the standard color variables listed in the table with a single function call. This function has one argument: *colorset*. The *colorset* argument is a bit-mask used to indicate the desired grays and hues. See “Color set definitions” on page 171.

Grays and hues are requested in groups. Colors are allocated LUT slot numbers in the order the mask bits and colors are presented in the table. The ordering of the bits in the *colorset* parameter does not matter. Colors continue to be assigned as long as slots are still available. For example, SetLut(GRAYIBM | HUERGB | HUECMY) would result in the following LUT slot allocations:

slot 0 = black

slot 1 = gray33

slot 2 = gray67

slot 3 = white

slot 4 = blue

slot 5 = green

slot 6 = red

slot 7 = cyan

slot 8 = magenta

slot 9 = yellow

The color black is always supported, regardless of the format. The color black is provided automatically whenever the SetLut() function is called - no matter what argument is given. The colors foreground and background are automatically set to white and black whenever SetVersions(1) is called. These colors should not be changed by the user.

You should only request those colors that you actually plan to use. This is important because the LUT management function might otherwise drop colors that you really need to make room for ones that you really don't.

Colors may be requested that cannot be supported. If a color is requested that cannot be supported (given the current format), then the variable associated with that color will be set to zero. The variable grayscale is 0 when the 256 color linear gray scale is not supported.

When driving an 8-bit analog or 8-bit B/W digital display, a complete gray scale is at your disposal. This can be set up using the GRAYALL color mask bit. In this case, the numbers 0 to 255 can be passed to the RepPixel(index) function in order to obtain a gray value for use with the Transparent(color1) and Opaque(color1,color0) functions. The number 0 represents black and the number levelmax white. The numbers between 0 and levelmax will produce varying levels of gray. For example, in order to draw with 42% gray (that is, levelmax*0.42), you could use the following code:

```
SetLut(GRAYALL);
/* Maybe use 42% gray if grayscale is available. */
if (grayscale) Transparent( RepPixel( Round
( 0.42*(double)GetLevelMax() ) ) );
```

The SetLut() function is significantly slower when using the GRAYALL color mask bit, especially when gamma correction is enabled.

Specific colors can be selected, one at a time by name, using the SetColorNamed() function as follows (warning: this method is slow and increases code size):

```
/* Only setup black, red, and blue. */
/* Set up the color black at slot 0 */
SetLut(BLACK);
/* Set up the color red at slot 1 */
Set_ColorNamed("red");
/* Set up the color blue at slot 2 */
SetColorNamed("blue");
```

You can select a predefined color for redefinition using the SelectColorNamed() function as follows:

```
/* Redefine the color named "white" to display blue */
/* Set up the colors black and white at slots 0 & 1 */
SetLut(BW);
SetColorNamed("white");
/* Display blue for "white" */
SetLevelRGB(0,0,GetLevelMax());
```

Color set definitions

The following table defines the colors in each color set.

Colorset	Colors
BW	black
	white
GRAYDEC	black
	gray10
	gray20
	gray30
	gray40
	gray50
	gray60
	gray70
	gray80
	gray90
	white

Colorset	Colors
GRAYHEX	black
	gray7
	gray13
	gray20
	gray27
	gray33
	gray40
	gray47
	gray53
	gray60
	gray67
	gray73
	gray80
	gray87
	gray93
	white

Colorset	Colors
GRAYTRI	black
	gray3
	gray7
	gray10
	gray13
	gray17
	gray20
	gray23
	gray27
	gray30
	gray33
	gray37
	gray40
	gray43
	gray47
	gray50
	gray53
	gray57
	gray60
	gray63
	gray67
	gray70
	gray73
	gray77
	gray80
	gray83
	gray87
	gray90
	gray93
	gray97
	white

Colorset	Colors
GRAYIBM	black
	gray33
	gray67
	white
GRAYACE	black
	gray30
	gray70
	white
GRAYCON	gray48
	gray50
	gray51
	gray53
GRAYEND	gray5
	gray95
GRAYMID	gray50
GRAYQTR	gray25
GRAYPEN	gray75
GRAYLOW	gray30
GRAYHIT	gray70
GRAYEXT	gray2
	gray4
	gray12
	gray35
	gray45
	gray55
	gray65
	gray85
HUERGB	red
	green
	blue

Colorset	Colors
HUECMY	cyan
	magenta
	yellow
HUEMID	blue50
	green50
	cyan50
	red50
	magenta50
	yellow50
HUEIBM	blue50
	green50
	cyan50
	red50
	magenta50
	yellow50
	blue
	green
	cyan
	red
	magenta
	yellow
HUEPEN	cyan75
	green75
	magenta75
	red75
	blue75
HUEIQ	huel (R=1.0*0.4 G=0.4056*0.4 B=0.0)
	hueQ (R=0.5371*0.4 G=0.0 B=1.0*0.4)
	huenegI (R=0.0 G=0.5944*0.4 B=1.0*0.4)
	huenegQ (R=0.4629*0.4 G=1.0*0.4 B=0.0)

Colorset	Colors
HUEPULSE	Setup huepulse[] array with samples for sine-squared reference red burst pulse generation. Number of samples is stored in the first element huepulse[0]. The half-amplitude delay (HAD) is automatically set to either 12.5T (NTSC) or 20T/2000ns (PAL) based on the current format. Color of the huepulse is R=1.0, B=0.25, G=0.25.
GRAYPULSE	Setup graypulse[] array with samples for sine-squared pulse generation. Number of samples is stored in the first element graypulse[0]. The 2T half-amplitude delay (HAD) is automatically set to either 250ns (NTSC) or 200ns (PAL) based on the current format.
NOGAMMA	Load colors as if current format were not gamma corrected.
TOBLANK	Load colors as if current format had no blanking pedestal.

Color definitions

The following table provides the RGB values for pre-defined colors. A number in the color name refers to its intensity level as a percentage. Colors and gray levels cannot be modified.

Name	Red Level	Green Level	Blue Level
black	0	0	0
red	255	0	0
green	0	255	0
yellow	255	255	0
blue	0	0	255
magenta	255	0	255
cyan	0	255	255
white	255	255	255
red50	128	0	0
green50	0	128	0
yellow50	128	128	0
blue50	0	0	128
magenta50	128	0	128
cyan50	0	128	128
brown	128	128	0
gray2	5	5	5
gray3	8	8	8

Name	Red Level	Green Level	Blue Level
gray4	11	11	11
gray5	13	13	13
gray7	18	18	18
gray10	26	26	26
gray12	29	29	29
gray13	33	33	33
gray17	43	43	43
gray20	51	51	51
gray23	59	59	59
gray25	64	64	64
gray27	69	69	69
gray30	77	77	77
gray33	84	84	84
gray35	89	89	89
gray37	94	94	94
gray40	102	102	102
gray43	110	110	110
gray45	115	115	115
gray47	117	117	117
gray48	122	122	122
gray50	128	128	128
gray51	130	130	130
gray53	135	135	135
gray55	140	140	140
gray57	145	145	145
gray60	153	153	153
gray63	161	161	161
gray65	166	166	166
gray67	171	171	171
gray70	179	179	179
gray73	186	186	186
gray75	191	191	191
gray77	196	196	196
gray80	204	204	204

Name	Red Level	Green Level	Blue Level
gray83	212	212	212
gray85	217	217	217
gray87	222	222	222
gray90	230	230	230
gray93	237	237	237
gray95	242	242	242
gray97	247	247	247
red75	191	0	0
green75	0	191	0
yellow75	191	191	0
blue75	0	0	191
magenta75	191	0	191
cyan75	0	191	191
huel	102	41	0
hueQ	55	0	102
huenegI	0	61	102
huenegQ	47	102	0
grayminus2	-6	-6	-6
grayminus4	-10	-10	-10
foreground	255	255	255
background	0	0	0
black	0	0	0
background	0	0	0
background	0	0	0

SetOrigin

Class Windows

Description Locates the x-y origin for the current viewport at the specified position, given in terms of displacements (x, y) from the top left corner of the viewport. The clipping rectangle, which is specified relative to the origin, is automatically adjusted to follow the change in position of the origin.

Syntax SetOrigin(int32 x, int32 y)

x, y
Coordinates of the origin for the current viewport.

Example This example draws an image that has different locations of coordinate arrows.

```
static short ptlist[] = {
            0,-10, 0,70, -4,62, 4,62,
           -10,0, 70,0, 62,-4, 62,4
      };
      static short axes[] = {
         0,1, 1,2, 1,3, 4,5, 5,6, 5,7
      };
      int i, x, y;
      int  ViewA,  ViewB;

      /* If changing the origin it is better to use a temporary
      viewport */
      ViewA = CurrentVuport();
      ViewB = OpenVuport();
      SelectVuport(ViewB);

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      Transparent(white);
      /* Move origin to various positions on screen */
      for (x = 10; x < 639; x += 100)
         for (y = 1; y < 479; y += 100) {
         SetOrigin(x, y);
         DrawPolyLine(6, axes, ptlist);
      }
      SelectVuport(ViewA);
      CloseVuport(ViewB);
```

SetVersions

Class Image version

Description Sets quantity of image versions (all images must call this function).

Syntax SetVersions(int n)

n

Number of image versions.

0 = Image STEP key does nothing.

1 = Image STEP key toggles the imageVersion canvas variable between 1 & 0. Rendering code should render an inverted image whenever imageVersion==1 (NOTE: background & foreground color variables are automatically exchanged in set_lut() - invertable images merely need to execute versions(1) before set_lut() is called).

N>1 = Image STEP key toggles image sub-list feature on and off. Normally, when the image knob is rotated, entirely different image rendering functions are called. When the image sub-list feature is enabled, the image knob instead recalls the current image rendering function repeatedly, while varying imageVersion. Versions are requested via re-rendering the current image with the imageVersion canvas variable set to values in the range 0 to (N-1).

Example This example draws an image that has a solid filled rectangle at the upper left top corner.

```
/* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      Transparent(blue);
      FillRect(100, 100, 0, 0);
```

SizeVuport

Class Windows

Description Changes the width and height of the selected viewport. The position of the top left corner of the viewport remains fixed, but the bottom left corner moves to accommodate the viewport's new dimensions.

Syntax int32 SizeVuport(int32 width, int32 height)

width
Width of the viewport.

height
Height of the viewport.

Example This example draws an image that has multiple colored rectangles.

```
int i, w, h;

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
      BlackFillVuport();

      i = 0;

      /* Show changes in size of viewport */
      for (w = 640, h = 480; w > 0; w -= 32, h -= 24) {
         SizeVuport(w, h);
         SetColor1(RepPixel(++i));
         FillRect(2000, 2000, -1000, -1000);
      }
```

SquareDX

Class 2-D methods

Description Scales machine units on the y axis into another machine units on the x axis.

Syntax int32 SquareDX(int32 pixheight)

pixheight
y coordinate to be scaled according to the x axis.

Example This example draws an image that has a red circle of radius equal to the height of the screen.

```
void Test_SquareDX( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the active color
  gc->Transparent( TStdColor::red );

  //*** Draw a circle of radius equals height of the screen
  gc->DrawOval( gc->SquareDX( gc->height ), gc->height - 1,
                gc->centerleft - (gc->SquareDX( gc->height )/2),
                gc->top );
}
```

SquareDY

Class 2-D methods

Description Scales machine units on the x axis into another machine units on the y axis.

Syntax Int32 SquareDY(int32 pixelwidth)

pixelwidth
x coordinate to be scaled according to the y axis.

Example This example drawsan image that has a red circle of radius equal to the half width at center of the screen.

```
void Test_SquareDY( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the active color
  gc->Transparent( TStdColor::red );

  //*** Draw a circle of radius equals height of the screen
  gc->DrawOval( gc->width/2, gc->SquareDY( gc->width/2 ),
                gc->centerleft - (gc->width/4),
                gc->centertop - gc->SquareDY( gc->width/4 ) );
}
```

Transform

Class 3-D methods

Description Uses a 4x4 transformation matrix to transform (rotate, scale and translate) a list of 3D vertices. The elements of the *matrix[]* (transformation matrix) and *vertlist[]* (vertex list) arrays are 32-bit fixed point values, with 16 bits to the right of the binary point.

Syntax Transform(int32 matrix[ ], int32 n, int32 vertlist[ ])

matrix[]
4x4 transformation matrix.

n
Number of (x,y,z) triples in *vertlist[]* array.

vertlist[]
List of vertices, each of which is represented by x, y, and z coordinate values. Since each vertex is specified as three coordinates, the *vertlist[]* array contains 3*n 32-bit fixed point numbers.

Example This example draws an image that has a triangle rotating around one of its vertices.

```
typedef long FIX;
      static FIX rotation[] = {0, 0, 0};
      static FIX xyz[] = { -120,-170,0, 120,-170,0, 0,0,0 };
      static short connect[] = {0, 1, 2};
      FIX matrix[16], verts[3*3];
      short xy[2][2*3];
      long angle;   /* angle should be long */
      unsigned short Which = 0;
      unsigned long frame = WaitFrame(0)+1;
      int  ViewA,  ViewB;

      /* If changing the origin it is better to use a temporary
      viewport */
      ViewA = CurrentVuport();
      ViewB = OpenVuport();
      SelectVuport(ViewB);

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();

      /* Clear the screen with black. */
```

```
BlackFillVuport();

/* Draw a rotating triangle */
SetOrigin(320, 240);
for(;;){
   if(kbhit())break;
   for(angle = 0; angle < 360; ++angle) {
      if(kbhit())break;
      InitMatrix(matrix);
      rotation[0] = angle << 16L;
      Rotate(matrix, rotation);
      Int32ToFix(3*3, xyz, verts);
      Transform(matrix, 3, verts);
      VertexToPoint(3, verts, xy[Which]);
      Which = Which ^ 1;
      WaitFrame(frame);
      frame+=1;
      Opaque(black,black);
      EncloseFillRect(3, xy[Which]);
      Which = Which ^ 1;
      Transparent(red);
      FillConvex(3, connect, xy[Which]);
      Which = Which ^ 1;
   }
}
SelectVuport(ViewA);
CloseVuport(ViewB);
```

TransformDX

Class 2-D methods

Description Transforms a normalized distance into an equivalent distance in machine units along the x axis. The variable xnormdist is normalized to 1.0 for the minor axis.

Syntax int32 TransformDX(double dx)

dx
Normalized distance to be returned as a machine unit.

Example This example draws an image that has a point at top side and halfway right + 10 machine units.

```
void Test_TransformDX( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the active color
  gc->Transparent( TStdColor::white );

  gc->ResetMatrix();

  //*** Draw a point at top middle + 10 machine units
  gc->DrawPoint( (gc->TransformDX( gc->normalwidth/2 ) + 10),
  0 );
}
```

TransformDY

Class 2-D methods

Description Transforms a normalized distance into an equivalent distance in machine units along y axis. The variable ynormdist is normalized to 1.0 for the minor axis.

Syntax int32 TransformDY(double dy)

dy
Normalized distance to be returned as a machine unit.

Example This example draws an image that has a point at left side and halfway down + 10 machine units.

```
void Test_TransformDY( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the active color
  gc->Transparent( TStdColor::white );

  gc->ResetMatrix();

  //*** Draw a point at left side and halfway down + 10 machine units
  gc->DrawPoint( 0, (gc->TransformDY( gc->normalheight/2 ) + 10)
  );
}
```

TransformX

Class 2-D methods

Description Transforms a normalized coordinate into an equivalent machine unit on the x axis. The variable xnormcoord is normalized to 1.0 for the minor axis.

Syntax int32 TransformX(double x)

x
x coordinate to be transformed into machine units.

Return value This function returns long.

Example This example draws an image that has 2 red points at top of screen in current normalized axis coordinates.

```
void Test_TransformX( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the active color
  gc->Transparent( TStdColor::red );

  gc->ResetMatrix();

  //*** Draw a point at top of screen and the x coordinate in current
  //*** normalized axis coordinates
  gc->DrawPoint( gc->TransformX(0.75) ,0);

  //*** Draw a point at top of screen and where x=0.0 in current
normalized
  //*** axis coordinates.
  gc->DrawPoint( gc->TransformX(0.0) , 0);
}
```

TransformY

Class 2-D methods

Description Transforms a normalized coordinate into an equivalent machine unit on the y axis. The variable ynormcoord is normalized to 1.0 for the minor axis.

Syntax int32 TransformY(double y)

double y
y coordinate to be transformed into machine units.

Return value This function returns long.

Example This example draws an image that has 2 white points 1 at top left of screen and the other at left of screen and the y coordinate in current normalized axis coordinates.

```
void Test_TransformY( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** Set the active color
  gc->Transparent( TStdColor::white );

  gc->ResetMatrix();

  //*** Draw a point at left of screen and the y coordinate in current
  //*** normalized axis coordinates
  gc->DrawPoint( 0, gc->TransformY(0.75) );

  //*** Draw a point at top of screen and where y=0.0 in current
normalized
  //*** axis coordinates
  gc->DrawPoint( 0, gc->TransformY(0.0) );
}
```

Translate

Class 3-D methods

Description Multiplies the specified 4x4 transformation matrix by a translation matrix created from the three displacements given as input arguments. The displacements in the x, y and z directions are stored in disp_val[0], disp_val[1] and disp_val[2], respectively. The elements of both the *matrix[]* (transformation matrix) and *disp_val[]* (x, y and z displacements) arrays are in fixed point format.

The matrix is defined as:

1 0 0 0

0 1 0 0

0 0 1 0

Tx Ty Tz 1

Syntax Translate(int32 matrix[ ], int32 disp_val[ ])

matrix[]

disp_val[]

Example This example draws an image that has a drawing of a hand watch graphic.

```
typedef long FIX;
      static FIX rotation[3] = {0, 0, 0};
      static FIX translat1[3] = {-320, -240, 0};
      static FIX translat2[3] = {320, 240, 0};
      static long xyz[] = {
         320,40,0, 340,240,0, 320,260,0, 300,240,0
      };
      static short connect[4] = {0, 1, 2, 3};
      FIX   matrix[16];
      FIX   verts[12];
      short xy[2][8];
      unsigned short Which = 0;
      long   angle;      /* angle should be long */
      int  ViewA,  ViewB;

      /* If changing the origin it is better to use a temporary
      viewport */
      ViewA = CurrentVuport();
      ViewB = OpenVuport();
      SelectVuport(ViewB);
```

```
/* Establish number of image versions. */
SetVersions(0);

/* Setup look-up-table for colors white and black and colors red,
green and blue */
SetLut(BW|HUERGB);

init_canvas();

/* Clear the screen with black. */
BlackFillVuport();

/* Draw a hand watch */
Int32ToFix(3, translat1, translat1);
Int32ToFix(3, translat2, translat2);
for(;;){
   if(kbhit())break;
   for (angle = 0; angle < 360; ++angle) {
      if(kbhit())break;
      InitMatrix(matrix);
      Translate(matrix, translat1);
      rotation[0] = angle << 16L;
      Rotate(matrix, rotation);
      Translate(matrix, translat2);
      Int32ToFix(12, xyz, verts);
      Transform(matrix, 4, verts);
      VertexToPoint(4, verts, xy[Which]);
      Which = Which ^ 1;
      WaitFrames(5);
      Opaque(black,black);
      EncloseFillRect(4, xy[Which]);
      Which = Which ^ 1;
      Transparent(red);
      DrawOval(420, 420, 110, 30);
      Transparent(blue);
      FillConvex(4, connect, xy[Which]);
      Which = Which ^ 1;
   }
}
SelectVuport(ViewA);
CloseVuport(ViewB);
```

TranslateX

Class 2-D methods

Description Translates the origin used by the transform and translate functions. A positive value translates to the right and a negative value translates to the left. The origin is translated by xnormdist which is normalized to 1.0 for the minor axis. Translations compound to translate from the latest axis. The default value is the left-hand side of screen set by ResetMatrix.

Syntax TranslateX(double x)

x
Normalized distance to translate axis to right.

Example This example demonstrates the TranslateX() function. No image is produced. The y axis is translated.

```
ResetMatrix();
      /* Translate y axis to x=0.25 or 0.25 length of minor axis */
      TranslateX(0.25);
      ResetMatrix();
      /* Translate y axis to middle of screen or normalwidth/2 after
      resetting matrix to defaults */
      TranslateX(normalwidth/2);
      /* Translate y axis to right edge with another half screen
      translation */
      TranslateX(normalwidth/2);
```

TranslateY

Class 2-D methods

Description Translates the origin used by the transform and translate functions. A positive value translates downward and a negative value translates upward. The origin is translated by ynormdist which is normalized to 1.0 for the minor axis. Translations compound to translate from the latest axis. The default value is the top of screen set by ResetMatrix.

Syntax TranslateY(double y)

y
Normalized distance to translate axis down.

Example This example demonstrates the TranslateY() function. No image is produced. The x axis is translated.

```
ResetMatrix();
      /* Translate x axis to y=0.25 or 0.25 length of minor axis */
      TranslateY(0.25);
      ResetMatrix();
      /* Translate x axis to bottom of screen or normalheight after
      resetting matrix to defaults */
      TranslateY(normalheight);
      /* Translate x axis back up to middle of screen */
      TranslateY(-normalheight/2);
```

Transparent

Class Transparency

Description Sets the foreground color. It also sets the drawing replacement mode. The Transparent() function will draw only the foreground of text and patterns unaltering where background color would be. The Transparent() function should be used in most instances.

Syntax void Transparent(uint32 color)

color
Foreground color value.

Example This example draws an image with "Black" text on a cyan rectangle in black background.

```
void Test_Transparent( TGC* gc )
{
  //*** set up 50% intensity colors and black                    and white
  gc->SetLut( BW | HUEMID );

  //*** Clear the screen
  gc->BlackFillVuport();

  //*** set up 50% intensity colors and black                    and white
  if( TStdColor::cyan50 )
  {
    gc->Transparent( TStdColor::cyan50 );

    // Fill a rectangle in 50% intensity cyan
    gc->FillRect( 200, 200, 300, 300 );
  }

  gc->Transparent( TStdColor::black );

  gc->SelectFont( "BIG30" );

  //*** Draw "Black" on cyan rectangle in black background is transparent
so
  //*** border of characters unaltered.
  gc->DrawText( 350, 350, "Black" );
}
```

TranspON

Class Register access

Description Clears the T (transparency) bit in the generator's rendering color registers. When transparency is enabled, and the result of a pixel operation involving the source and destination pixels is zero, the destination pixel is not altered.

Syntax TranspON(void)

Example This example uses the TranspOFF and TranspON functions to draw an image with overlapping rectangles and pattern filled ovals.

```
void Test_TranspOFF( TGC* gc )
{
  //*** Setup look-up-table for colors white and black and red green blue
  gc->SetLut( BW | HUERGB );

  //*** Clear the screen
  gc->BlackFillVuport();

  gc->SelectPatn( "CHECKER8" );
  gc->Opaque( TStdColor::green, TStdColor::black );
  gc->PatnFrameOval( 150, 150, 45, 165, 35, 35 );

  //*** Construct 2 copies of destination
  gc->Transparent( TStdColor::red );
  gc->FillRect( 190, 190, 225, 145 );
  gc->Transparent(TStdColor::blue);
  gc->FillRect( 30, 190, 305, 145 );
  gc->FillRect( 190, 30, 225, 225 );
  gc->MoveRect( 200, 200, 220, 140, 420, 140 );

  //*** Copy source to 1st dest. with transp. true
  gc->TranspON();
  gc->MoveRect( 160, 160, 40, 160, 240, 160 );

  //*** Copy source to 2nd dest. with transp. false
  gc->TranspOFF();
  gc->MoveRect( 160, 160, 40, 160, 440, 160 );
}
```

TranspOFF

Class Register access

Description Disables transparency, and the result of a subsequent pixel operation is written to the destination regardless of its value.

Syntax TranspOFF(void)

Example See page 195.

UnRepPixel

Class Color

Description Unreplicates pixel throughout a 32-bit integer. Shifts the n MSBs of the input argument into the n LSBs of same. For example, given a pixel size of 4 and an input value of 0x55555555, the return value would be 0x00000005. The output of this function is in suitable form to be used as input to the SelectColor(), and RepPixel() functions.

Syntax uint32 UnRepPixel(uint32)

Example

VertexToPoint

Class Fixed point methods

Description Converts a vertex list to a point list by copying the integer parts of the x and y coordinates from the vertex list to the point list.

Syntax VertexToPoint(int32 n, int32 vertlist[ ], TQD_Point ptlist[ ])

n

Number of vertices in *vertlist[]* and the number of points in *ptlist[]*.

vertlist[]

List of three-dimensional vertices, each of which is a triple (x,y,z). The x, y and z coordinates for each vertex are 32-bit fixed point numbers, with 16 bits to the right of the binary point. The number of 32-bit fixed-point coordinate values in the vertex list is calculated as 3*nverts.

ptlist[]

List of two-dimensional points, each of which is expressed as a coordinate pair (x,y). The x and y coordinates are in the form of 16-bit integers. The number of 16-bit short coordinate values in the point list is calculated as 2*nverts.

Example This example draws an image that has a pattern filled convex shape.

```
typedef long FIX;
      static  long xyz[] = {
         0,-100,0, 100,0,0, 0,100,0, -100,0,0
      };
      static short diamond[] = {0, 1, 2, 3};
      FIX   verts[12];
      short xy[8];
      int  ViewA,  ViewB;

      /* If changing the origin it is better to use a temporary
      viewport */
      ViewA = CurrentVuport();
      ViewB = OpenVuport();
      SelectVuport(ViewB);

      /* Establish number of image versions. */
      SetVersions(0);

      /* Setup look-up-table for colors white and black and colors red,
      green and blue */
      SetLut(BW|HUERGB);

      init_canvas();
```

```
/* Clear the screen with black. */
BlackFillVuport();

SetOrigin(320, 240);
Int32ToFix(12, xyz, verts);
VertexToPoint(4, verts, xy);
SelectPatn(CHECKER8);
Opaque(white,blue);
PatnFillConvex(4, diamond, xy);
SelectVuport(ViewA);
CloseVuport(ViewB);
```

ZoomRect

Class Bitmap icons

Description Expands or shrinks a source rectangle on the screen to fit dimensions of a destination rectangle that is on the screen. This type of function is sometimes referred to as a "stretch blit." Horizontal zooming is done by replicating (if expanding) or eliminating (if shrinking) columns of pixels from the source rectangle. Vertical zooming is done by replicating (if expanding) or eliminating (if shrinking) rows of pixels from the source rectangle. When shrinking, several pixels from the source rectangle are collapsed into a single pixel in the destination rectangle. The source pixels collapsed in this manner are combined according to the currently selected pixel processing operation. For example, the replace operation simply selects a single source pixel to represent all the pixels in the region being collapsed. A better result can often be obtained using a logical-OR (at one bit per pixel) or MAX (at multiple bits per pixel) operation instead. Both the source and destination rectangles are on the screen.

Syntax ZoomRect(int32 srcwidth, int32 srcheight, int32 srcx, int32 srcy, int32 destwidth, int32 destheight, int32 destx, int32 desty, int32 buffer[ ])

srcwidth
Width of the source rectangle.

srcheight
Height of the source rectangle.

srcx, srcy
Coordinates of the top left corner of the source rectangle.

destwidth
Width of the destination rectangle.

destheight
Height of the destination rectangle.

destx, desty
Coordinates of the top left corner of the destination rectangle.

buffer[]
A buffer large enough to contain one full scan line of the display. For example, if the screen is 640 pixels wide, and each pixel is 4 bits, the buffer capacity must be at least 640x4 = 2560 bits.

Example This example draws an image that has zoomed filled rectangles.

```
/* Establish number of image versions. */
      SetVersions(0);
```

```
/* Setup look-up-table for colors white and black and colors red,
green and blue */
SetLut(BW|HUERGB);

init_canvas();

/* Clear the screen with black. */
BlackFillVuport();

Opaque(red,black);
DrawRect(63, 63, 0, 0);

ZoomRect(64, 64, 0, 0, 200, 200, 120, 40, NULL);

Transparent(blue);
SeedFill(125, 45, NULL, 1000);

ZoomRect(200, 200, 120, 40, 100, 100, 300, 300, NULL);

Opaque(red,white);
SelectPatn(CHECKER8);

SeedPatnFill(305, 305, NULL, 1000);
```

3 Image Examples

Topic in this chapter:

- PulseBar
- SMPTEBar
- QuartBox
- BurstTCE
- Diamond
- Samples

PulseBar

This image has two vertical lines followed by a wide vertical bar on a display's screen. The first line is a sine-squared modulated pulse that fades from black to red and back to black. The pulse is 20 T for PAL and 12.5 T for NTSC formats. The second narrower line is a 2 T white sine-squared pulse. T = 100 nSec for PAL and 125 nSec for NTSC formats. The wide bar is white with sine-squared edges.

The source code for this image is at \sdk\examples\barpulse.cpp.

DISTRIBUTED UNDER NON-DISCLOSURE AGREEMENT

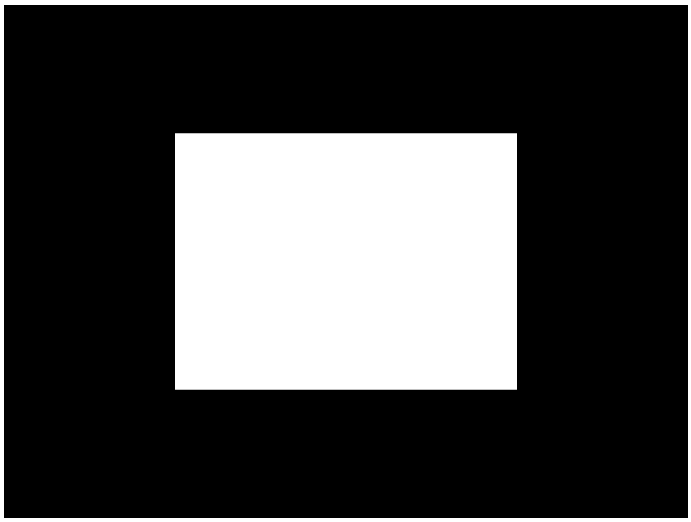
SMPTEBar

This image is based on an engineering guideline (EG1-1990) test signal specified by the Society of Motion Picture and Television Engineers (SMPTE). It is designed for adjusting the color settings of a television monitor by eye.

The source code for this image is at \sdk\examples\barsmpte.cpp.

QuartBox

The primary version (shown below) has a single white box in the center of active video. The size of the box is one-half the width and height of the active video area (a quarter of the entire active video area). The secondary version draws a black box on a white background.

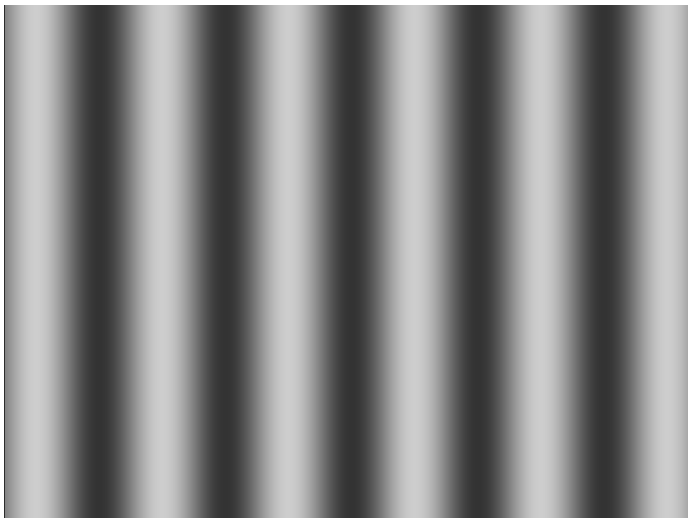


The source code for this image is at \sdk\examples\boxquart.cpp.

DISTRIBUTED UNDER NON-DISCLOSURE AGREEMENT

BurstTCE

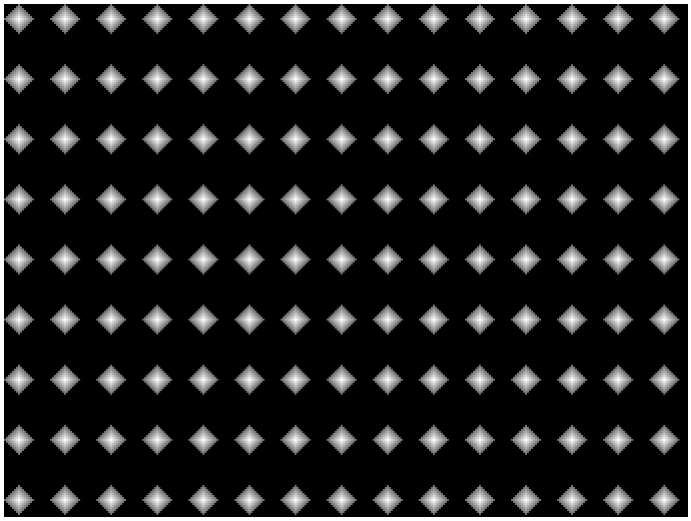
Fills screen with a 0.5 MHz frequency. There are multiple versions for this image.



The source code for this image is at \sdk\examples\tceburst.cpp.

Diamond

The diamond image is a special test image developed per customer specifications. It is created by duplicating the same diamond primitive across and down the screen.



The source code for this image is at \sdk\examples\diamond.cpp.

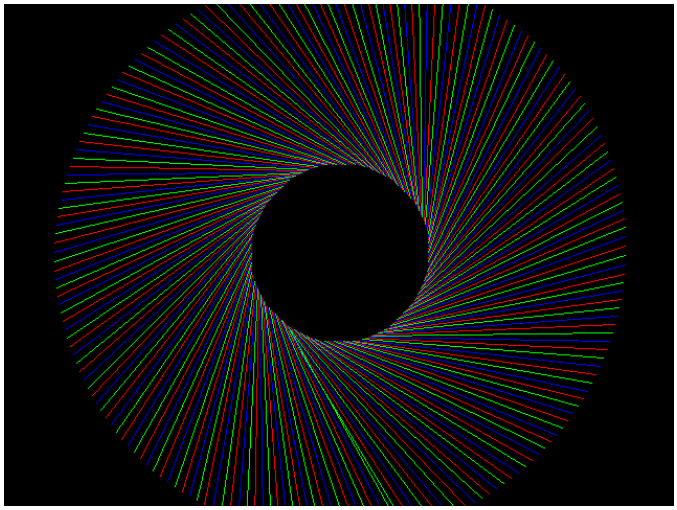
DISTRIBUTED UNDER NON-DISCLOSURE AGREEMENT

Samples

This image has 77 versions. Each version demonstrates one or more functions of the SDK.

To load the samples image:

1. Put the generator in Basic operations mode.
2. Press the Content key.
3. Choose the Samples image.
4. Press the Options keys to view the image options.
5. Press the **+**key to progress through the images.
6. The version is indicated by the count next to the Rendition field.



The source code for this image is at \sdk\examples\samples.cpp.

DISTRIBUTED UNDER NON-DISCLOSURE AGREEMENT